

JC630 U.S. PTO  
09/24/99  
01/29/99

ASSISTANT COMMISSIONER OF PATENTS  
Washington, D.C. 20231

CASE DOCKET NO. Y0999-014

Date: January 29, 1999

Express Mail Label #EL295371866US

Date of Deposit: January 29, 1999

Transmitted herewith for filing under Rule 53(b) is the Patent Application of:

Inventors: Pravin Bhagwat and John M. Tracey

For: IMPROVING PERFORMANCE OF INTERMEDIATE NODES WITH FLOW SPLICING

Enclosed are:

X 11 sheets of drawing(s).

     An assignment of the invention to International Business Machines Corporation.

     Declaration and Power of Attorney.

     A certified copy of a    application.

The filing fee has been calculated as shown below:

(Col. 1)		(Col. 2)	OTHER THAN A SMALL ENTITY	
FOR:	NO. FILED	NO. EXTRA	RATE	FEE
BASIC FEE				\$760.00
TOTAL CLAIMS	58-20 =	38	X \$18 =	684.00
INDEP CLAIMS	4- 3 =	1	X \$78 =	78.00
<u>    </u> MULTIPLE DEPENDENT CLAIM PRESENTED			+ 260 =	
If the difference in Col. 1 is less than zero, enter "0" in Col. 2.			SUBTOTAL	\$1522.00
				\$
			TOTAL	\$1522.00

X Please charge my Deposit Account No. 09-0468 in the amount of \$1522.00.

X The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 09-0468. A duplicate copy of this sheet is enclosed.

  x   Any additional filing fees required under 37 CFR 1.16.

  x   Any patent application processing fees under 35 CFR 1.17.

Respectfully submitted,

By Louis Herzberg  
Attorney: Louis P. Herzberg  
Registration No.: 41,500  
Tel. (914) 945-2885

IBM CORPORATION  
INTELLECTUAL PROPERTY LAW DEPT.  
P.O. BOX 218  
YORKTOWN HEIGHTS, NY 10598

JC530 U.S. PTO  
09/24/99  
01/29/99

1     Improving Performance of Intermediate Nodes  
2                     with Flow Splicing

3     Field of The Invention

4     The invention is directed to communication via  
5     packet switched networks. It more particularly is  
6     directed to methods for improving the performance of  
7     intermediate network nodes that participate at or above  
8     the transport layer in communication between one or  
9     more node pairs.

10    Background of the Invention

11    Traditionally, internet software has been developed  
12    in accordance with the principle that very limited  
13    functionality is provided in "the network."  
14    Specifically the network provides unreliable forwarding  
15    of messages or *packets*. Following this model,  
16    additional functionality, such as reliable delivery and  
17    flow control, is implemented entirely at communication  
18    end points without adding any functionality to the  
19    network.

20    Figure 1 depicts a TCP connection between a first  
21    node 101 and a second node 102 following this model.  
22    The connection extends "directly" from the first node  
23    101 to the second node 102 with only IP routers 103 in  
24    between. Routers do not maintain state information  
25    associated with constructs that reside above the

1 network layer. Connection state, which is associated  
2 with the transport and session layer, is therefore  
3 maintained solely in the two connection end points,  
4 that is at the first node 101 and second node 102.

5 As internet technology has evolved, significant  
6 functionality has found its way into "the network."  
7 For example, it is increasingly common for a client  
8 running a web browser to connect to an *intermediate*  
9 *node* rather than directly to a web server. Such  
10 intermediate nodes include but are not limited to:  
11 SOCKS servers, fire walls, VPN (virtual private  
12 network) gateways, TCP routers or load balancers,  
13 caching proxies and transcoding proxies used with  
14 resource-constrained clients such as handheld devices.

15 The number and types of such intermediate nodes will  
16 continue to increase as internet technology evolves to  
17 provide additional functionality. The performance of  
18 the intermediate nodes will increase in importance as  
19 the number of internet clients continues to grow  
20 rapidly. This growth will be fueled by large numbers  
21 of handheld and pervasive devices that will require  
22 intermediate nodes capable of supporting hundreds of  
23 thousands to millions of clients simultaneously.

24 Figure 2 depicts this increasingly common scenario  
25 in which a first node 201 connects to a second node 202  
26 via a set of intermediate nodes 203. Although not  
27 explicitly depicted in the figure, intermediate nodes  
28 typically communicate with each other as well as with  
29 the first node and the second node via routers.

30 Each intermediate node influences communication  
31 between the first node 201 and the second node 202. In  
32 many instances, the interface provided by an

1 intermediate node to the first node 201 resembles or is  
2 identical to the interface provided by the second node  
3 202 to the intermediate node. Similarly, the interface  
4 provided by an intermediate node to the second node 202  
5 is typically similar to the interface provided by the  
6 first node 201 to the intermediate node. Because of  
7 this similarity of interfaces, the first node may not  
8 be able to determine if is connected to an intermediate  
9 node or to the second node. Similarly there may be no  
10 way for the second node to determine if it is connected  
11 to an intermediate node or to the first node. The  
12 similarity of interfaces allows there to be any number  
13 of intermediate nodes between the first node 201 and  
14 the second node 202. This property is depicted in  
15 Figure 3.

16 A key difference between a router and an  
17 intermediate node is that routers perform processing  
18 only at layers one through three of the ISO seven layer  
19 model, those are the *physical*, *data link* and *network*  
20 layers, while intermediate nodes perform processing at  
21 and possibly above the fourth or *transport* layer.  
22 Figure 4 depicts the processing performed by a router  
23 in terms of layers. The corresponding diagram for  
24 intermediate nodes is shown in Figure 5.

25 We define a *connection* to be a bi-directional  
26 communication channel between a pair of connection end  
27 points such that information written to one end of the  
28 connection can be read from the other end. A  
29 connection includes two *flows*. A flow is a  
30 unidirectional communication channel between a pair of  
31 flow end points such that information written to the  
32 source flow end point can subsequently be read from the

1 corresponding destination flow end point. A connection  
2 need not necessarily be supported by a connection  
3 oriented protocol. All that is required is the  
4 identification of a pair of connection end points and  
5 propagation of data between the end points.  
6 Connections and flows reside at the fifth or *session*  
7 layer in the ISO model. Because routers in general do  
8 not perform processing above layer three, routers  
9 generally do not perform processing explicitly related  
10 to connections. Intermediate nodes however do  
11 generally perform processing explicitly associated with  
12 connections.

13 In fact, intermediate nodes can be distinguished  
14 from routers in terms of connections. In Figure 1, the  
15 first node 101 communicates with the second node 102  
16 via a single connection 104 whereas in Figure 2  
17 communication between the two nodes takes place via  
18 multiple connections in series 204, 205, 206, 207, 208  
19 and 209. The use of multiple connections provides each  
20 intermediate node with end points that can be used to  
21 influence communication between the first node 201 and  
22 the second node 202. However, intermediate nodes  
23 typically expend much of their resources simply moving  
24 data from one connection to another. In one common  
25 scenario, the intermediate node monitors the flow of  
26 information between the first node 201 and second node  
27 202 only until a request made by the first node 201 can  
28 be identified. In another common scenario, the  
29 intermediate node monitors the flow of information only  
30 in one direction. The performance and capacity of an  
31 intermediate node is often determined therefore by the

1 efficiency with which it moves data between  
2 connections.

3 Having a series of connections between the first and  
4 second node can cause several undesirable side-effects.  
5 Connections in series tends to deliver worse  
6 performance in terms of latency compared to a single  
7 connection and may also degrade throughput. Each node  
8 in a packet switched network introduces some delay and  
9 imposes a throughput limit. The performance of a  
10 packet switched network therefore relies on minimizing  
11 the delay introduced and maximizing the throughput  
12 supported by each node. This is accomplished, in part,  
13 by performing only minimal processing at each node.  
14 The packet forwarding performed by a router entails  
15 only a small amount of overhead, but the processing  
16 associated with connection end points performed by an  
17 intermediate node is significant.

18 The presence of multiple connections also alters the  
19 semantics of communication between the first node 201  
20 and the second node 202 of Figure 2. For example, with  
21 a single connection, the first node 201 is assured data  
22 has arrived at the second node 202 when it receives an  
23 *acknowledgment*. With multiple connections, the first  
24 node 201 may be led to believe data has arrived at the  
25 second node 202 when, in fact, it has only reached the  
26 first intermediate node.

## 27 Summary of the Invention

28 It is therefore an object of the present invention  
29 to improve the performance of intermediate nodes using

1 an approach called *flow splicing*. A flow splice  
2 transforms a pair of flows, a first flow inbound to an  
3 intermediate node and a second flow outbound from the  
4 intermediate node, into a single third composite flow  
5 that originates at the source of the first flow and  
6 terminates at the destination of the second flow. The  
7 transformation is invisible from the source node at  
8 which the first flow originated and the destination  
9 node at which the second flow terminated.

10 A flow splice is applied to a pair of flows in one  
11 direction between a first and second node independent  
12 of any associated flows. For example, a flow splice  
13 transforms one flow of a connection without modifying  
14 the other flow associated with the same connection.  
15 The unidirectional nature of a flow splice is an  
16 essential characteristic as techniques used to splice  
17 connections do not generally allow data flow to be  
18 spliced in one direction only. The ability to splice in  
19 one direction only greatly increases the number of  
20 scenarios in which splicing can be applied.

21 Figure 6 depicts a flow splice. In this figure, a  
22 splice is performed at the intermediate node I 603  
23 between the inbound flow from a first node 601 and the  
24 outbound flow to a second node 602. Flow splicing  
25 significantly improves network performance between  
26 nodes communicating via one or more network  
27 intermediaries and increase the capacity of the  
28 intermediaries.

## 29 Brief Description of Drawings

1 The foregoing and other objects, aspects and  
2 advantages of the invention may be better understood by  
3 referring to the following detailed description of a  
4 preferred embodiment with reference to the accompanying  
5 drawings, in which:

6 Figure 1 is a diagram of two nodes communicating via  
7 routers;

8 Figure 2 is a diagram of two nodes communicating via  
9 intermediate nodes;

10 Figure 3 is a conceptual diagram that depicts  
11 interfaces exported and bound to by intermediate nodes;

12 Figure 4 is a diagram that depicts layer processing  
13 performed by routers;

14 Figure 5 is a diagram that depicts layer processing  
15 performed by intermediate nodes;

16 Figure 6 is a diagram that depicts flow  
17 transformation resulting from a flow splice in  
18 accordance with the present invention;

19 Figure 7 is a schematic diagram of two nodes  
20 communicating via a connection;

21 Figure 8 is a schematic diagram of two nodes each of  
22 which has established a connection with an intermediate  
23 node;

24 Figure 9 is a schematic diagram that depicts the  
25 effect of using a flow splice to transform two flows  
26 into a single flow in accordance with the present  
27 invention;

28 Figure 10 is a schematic diagram that depicts the  
29 effect of using a flow splice to transform two flows  
30 into a single flow in accordance with the present  
31 invention;



1 Figure 11 is a schematic diagram that depicts the  
2 effect of using a pair of flow splices to transform two  
3 connections into a single connection.

4 Figure 12 is a schematic diagram that depicts the  
5 effect of applying two splices to transform six flows  
6 into four in a scenario in which three nodes have each  
7 established a connection with a fourth node in  
8 accordance with the present invention;

9 Figure 13 is a diagram of TCP header format;

10 Figure 14 is a state diagram that shows various  
11 states for a TCP to TCP splice and the logic that  
12 governs state transitions in accordance with the  
13 present invention;

14 Figure 15 is a flow diagram that provides an  
15 overview of the logic for processing TCP segments  
16 associated with a flow splice in accordance with the  
17 present invention;

18 Figure 16 is a flow diagram that describes splice  
19 destination processing in accordance with the present  
20 invention;

21 Figure 17 is a flow diagram that describes splice  
22 source processing in accordance with the present  
23 invention;

24 Figure 18 is a graphical representation of a  
25 sequence of packets flowing through an intermediate  
26 node without flow splicing;

27 Figure 19 is a graphical representation of a  
28 sequence of packets flowing through an intermediate  
29 node with a flow splice in accordance with the present  
30 invention.

## 1 Detailed Description of the Invention

### 2 **Splice Architecture**

3 A node is any device capable of communicating via  
4 one or more communication networks or links. Figure 7  
5 depicts two nodes A 701 and B 702 communicating via a  
6 connection  $C_{AB}$  705. The connection  $C_{AB}$  705 is  
7 bi-directional and includes of two flows. Flow  $F_{AB}$  703  
8 carries data from node A 701 to node B 702. Similarly,  
9 data flows in the reverse direction, that is from node  
10 B 702 to node A 701 along flow  $F_{BA}$  704. A connection  
11 therefore includes of two flows, one in each direction,  
12 between a pair of nodes.

13 A connection may also be denoted by a pair of end  
14 point pairs. In Figure 7, connection  $C_{AB}$  705 extends  
15 between end point pair  $EP_{A1}$  706 at node A 701 and end  
16 point pair  $EP_{B1}$  707 at node B 702. End point  $EP_{A1}$  706  
17 denotes the flow outbound from node A 701 via  $F_{AB}$  703  
18 and the inbound flow to node A 701 via  $F_{BA}$  704.  
19 Similarly, end point pair  $EP_{B1}$  707 denotes the flow  
20 inbound to node B 702 via  $F_{AB}$  703 and the flow outbound  
21 from node B 702 via  $F_{BA}$  704. The order in which the  
22 nodes linked by a connection are listed in the

23 connection name is not significant. For example the  
24 name  $C_{BA}$  denotes exactly the same connection as  $C_{AB}$  705.

25 The nature of a flow is that data written to the  
26 source flow end point can subsequently be read from the  
27 destination flow end point. Although boths ends of a  
28 flow can reside on the same node, flows are commonly  
29 extended between nodes through the use of transport  
30 protocols such as TCP/IP, AppleTalk and NETBIOS.

1 One aspect of the present invention allows two  
2 flows, one inbound the other outbound relative to a  
3 given node, both flows being terminated at the given  
4 node, to be transformed into a single flow through the  
5 given node. The method of the present invention  
6 transforms the two flows into one flow which originates  
7 at the source of the original inbound flow and  
8 terminates at the destination of the original outbound  
9 flow. The method does not require any particular  
10 relationship between the two flows to be transformed  
11 other than that one flow be inbound the other flow be  
12 outbound at the intermediate node at which the  
13 transformation is performed. The two flows may or may  
14 not be associated with the same connection. The source  
15 of the inbound flow and destination of the outbound  
16 flow may reside on the same node or on different nodes.  
17 The term "flow splice" or simply "splice" is used  
18 herein to refer to this described transformation.  
19 Before two flows are spliced, any amount of data can  
20 be read from the inbound flow and written to the  
21 outbound flow. Subsequent to the creation of a splice,  
22 data arriving on the inbound flow is automatically sent  
23 on the outbound flow. That is, data is propagated to  
24 the outbound flow without any further action by the  
25 entity that invoked the splice operation. Except for  
26 performance considerations, a pair of spliced flows  
27 resembles a pair of flows where data is read from the  
28 inbound flow as it arrives and is immediately written  
29 to the outbound flow.  
30 The splice operation has a wide variety of  
31 applications. For example the connections depicted in  
32 Figure 8 could correspond to any of many different

1 transport protocols including IP, UDP, TCP, IPX, OSI,  
2 NETBIOS, AppleTalk, etc. Any of several application  
3 programming interfaces (APIs) could be used to create  
4 and manipulate the flows transformed by a splice.  
5 These include the Berkeley Socket Interface, the  
6 STREAMS interface, NETBEUI and IBM's Common Programming  
7 Interface for Communication (CPI-C). Both flows  
8 transformed by a given splice may be associated with  
9 the same transport protocol or each may be associated  
10 with a different protocol. Each protocol may be either  
11 connection oriented or connection less. We proceed to  
12 describe the method of the present invention in its  
13 most general form and subsequently describe specific  
14 instances of the invention's application in more  
15 detail.

16 The specific details of implementing the splice  
17 operation vary somewhat depending on the particular  
18 transport protocols involved. One characteristic  
19 feature of a splice implementations is however that it  
20 "short circuits" or reduces protocol processing at and  
21 above the transport layer. An obvious approach to  
22 reducing processing above the transport layer is for  
23 the splice implementation to simply modify each packet  
24 received on the inbound flow to make it suitable for  
25 transmission on the outbound flow and immediately send  
26 the packet.

27 Another aspect of the present invention defines an  
28 architecture for manipulating data flows that can be  
29 combined with an existing communication architecture.  
30 The only requirement of the existing architecture is  
31 that it allows the establishment of data flows.

1 Figure 8 depicts a scenario that includes three  
 2 nodes A 801, I 802, and B 803 and two connections C<sub>AI</sub>  
 3 808 and C<sub>BI</sub> 809. Node I 802 represents an *intermediate*  
 4 *node* between nodes A 801 and B 803. An intermediate  
 5 node is a node that influences communication between  
 6 one or more pairs of nodes that communicate with each  
 7 another by connecting to the intermediate node. There  
 8 are many different ways in which an intermediate node  
 9 can influence communication between node pairs. For  
 10 example, an intermediate node might simply enable  
 11 communication between two sets of nodes that would not  
 12 otherwise be able to communicate. An intermediate node  
 13 might enforce security restrictions by allowing each  
 14 node to communicate with only a specifically authorized  
 15 set of nodes. An intermediate node might also improve  
 16 the performance of communication between two sets of  
 17 nodes by *caching*, that is retaining a copy of, data  
 18 transferred between the sets of nodes and providing  
 19 such data to a node when it is requested eliminating  
 20 the need for the data to be resent by the node from  
 21 which it was originally obtained.

22 Figure 9 shows the scenario that results from  
 23 transforming the scenario in Figure 8 using a flow  
 24 splice. Specifically, Figure 9 shows the results of  
 25 transforming Figure 8 by splicing flow F<sub>AI</sub> 804 to flow  
 26 F<sub>IB</sub> 806 at node I 802. The splice transforms the two  
 27 flows, F<sub>AI</sub> 804 and F<sub>IB</sub> 806, into a single flow F<sub>AB</sub> 904.  
 28 This splice operation can be described in terms of end  
 29 point pairs at node I 802. Specifically, Figure 9  
 30 depicts the inbound flow associated with end point pair  
 31 EP<sub>I1</sub> 811 having been spliced to the outbound flow  
 32 associated with end point pair EP<sub>I2</sub> 812. A splice

1 operation is therefore performed on a pair of flows.  
2 One flow is inbound and the other flow is outbound  
3 relative to the intermediate node at which the splice  
4 is performed.

5 By combining two flows into one, the splice  
6 eliminates two flow end points. The two are the  
7 destination end point of the inbound flow, and the  
8 source end point of the outbound flow. Note two of the  
9 so-called end point pairs in Figure 9, specifically  
10  $EP_{I1}$  908 and  $EP_{I2}$  909 include only a single end point.  
11 Also note Figure 9 does not show any connections as  
12 previously defined. That is it does not show any flow  
13 pairs in opposite directions between the same pair of  
14 nodes. The splice depicted in Figure 9 therefore  
15 eliminates the connections  $C_{AI}$  808 and  $C_{BI}$  809 shown in  
16 Figure 8.

17 The splice depicted in Figure 9 changes the  
18 destination of the outbound flow associated with end  
19 point pair  $EP_{A1}$  810 on node A 801 and similarly changes  
20 the source of the inbound flow associated with end  
21 point pair  $EP_{B1}$  813 on node B 803. Despite this, the  
22 splice is transparent to both node A 901 and node B  
23 903. Nothing intrinsic to the splice allows an  
24 observer on node A 901 to detect that data sent  
25 outbound from end point pair  $EP_{A1}$  907 ends up at node B  
26 903 instead of node I 902. Similarly, nothing  
27 intrinsic to the splice allows an observer on node B to  
28 detect that data received on end point pair  $EP_{B1}$  910  
29 originates at node A 901 instead of node I 902.

30 The splice is, in fact, undetectable from node B 903  
31 unless either the data received on the inbound flow  
32 associated with end point pair  $EP_{B1}$  910 or the rate at

1 which it arrives identifies the sender. Similarly the  
2 splice can not be detected from node A 901 unless the  
3 rate at which data is received after being sent on the  
4 outbound flow associated with end point pair  $EP_{AI}$  907  
5 can be detected at node A 901 in such a way as to  
6 identify the receiver.

7 Figure 10 also depicts a scenario that results from  
8 transforming the scenario in Figure 8 with a flow  
9 splice. Figure 10 shows the result of splicing the  
10 inbound flow associated with end point pair  $EP_{I2}$  812  
11 with the outbound flow of end point pair  $EP_{I1}$  811.  
12 Splicing these two flows results in a single flow  $F_{BA}$   
13 1006 shown in Figure 10. Figures 9 and 10 together  
14 show that the flows *in either direction* between a pair  
15 of nodes communicating via an intermediate node can be  
16 transformed into a single flow via a splice.

17 Figure 11 shows the scenario that results from  
18 transforming the scenario in Figure 8 with two splices,  
19 those depicted in Figure 9 and Figure 10. One splice  
20 transforms flows  $F_{AI}$  804 and  $F_{IB}$  806 into flow  $F_{AB}$  1104.  
21 The other splice transforms flows  $F_{BI}$  807 and  $F_{IA}$  805  
22 into flow  $F_{BA}$  1105. Unlike the scenarios depicted in  
23 Figure 9 and Figure 10, the scenario in Figure 11  
24 includes a connection. The two splices transform  
25 connections  $C_{AI}$  808 and  $C_{BI}$  809 into connection  $C_{AB}$  1106.  
26 Thus just as a flow splice transforms two flows into  
27 one, a pair of flow splices can transform two  
28 connections into one.

29 More complicated examples of flow splicing are  
30 possible. Figure 12 shows an example with four nodes,  
31 A 1201, I 1202, B 1203, and C 1204 and four flows  $F_{AI}$   
32 1205,  $F_{CA}$  1206,  $F_{IB}$  1207 and  $F_{BC}$  1208. The scenario in

1 Figure 12 resulted from transforming a scenario with  
2 three connections, one from node I 1202 to each of  
3 nodes A 1201, B 1203, and C 1204 with two flow splices.  
4 Another likely scenario is the case where a series of  
5 flows from a source node to a destination through a set  
6 of network intermediaries are transformed into a single  
7 flow from the source to destination using a number of  
8 flow splices.

9 We now consider the use of flow splicing. The  
10 splice architecture allows a flow to be spliced after  
11 any amount of data, or no data, has been sent or  
12 received on the flow. The architecture also allows a  
13 spliced flow to revert to the unspliced state either  
14 due to termination of the inbound flow or because the  
15 splice was explicitly eliminated at the intermediate  
16 node. This flexibility allows flow splicing to be used  
17 in a variety of ways.

18 One potential use for flow splicing is *load*  
19 *balancing*. An intermediate node can use flow splicing  
20 to balance the load from a set of clients across a set  
21 of servers. An intermediate node can implement load  
22 balancing by establishing one or more flows with client  
23 nodes, establishing one or more flows with server nodes  
24 and splicing flows between the clients and servers.

25 Another use for flow splicing is *content*  
26 *partitioning*. This refers to dividing a set of content  
27 (or services) across a set of servers possibly with  
28 multiple servers providing the same content. An  
29 intermediate node can make it appear to clients that  
30 all the content is available from a single server node.  
31 A content partitioning intermediate node operates



1 similarly to a load balancing intermediate node in  
2 terms of splicing flows between clients and servers.

3 As an example we consider an intermediate node that  
4 allows World Wide Web content to be partitioned across  
5 a set of servers. The web intermediate node starts by  
6 accepting a TCP connection from a client. It then  
7 reads one or more requests from the flow inbound from  
8 the client. Next the intermediate node determines an  
9 appropriate server to service the client request(s).  
10 It then establishes a connection, including of two  
11 flows one in either direction, with the selected server  
12 assuming an appropriate connection is not already  
13 available. The intermediate node then selects the  
14 inbound flow from the chosen server and the outbound  
15 flow to the client and splices the selected flows.  
16 Finally, the intermediate node writes the request that  
17 it had received from the client to the server.

18 The response from the server subsequently flows to  
19 the client via the splice. The client may close the  
20 connection after sending a request in which case  
21 processing for the connection is complete. The client  
22 might also send another request on the same connection  
23 in which case the intermediate node can either write  
24 the request to the same server allowing the response to  
25 flow across the established splice or resplice the  
26 outbound flow to the client to a different flow inbound  
27 from a server and send the client request on the  
28 corresponding flow outbound to that server instead.

29 It will be understood by those skilled in the art  
30 that the process just described can be applied to a  
31 wide variety of intermediate node functions and  
32 communication protocols. For example, the steps listed

1 above for the web intermediate node can be used to  
2 provide load balancing and content partitioning for a  
3 list of protocols including HTTP, SOCKS, telnet, FTP,  
4 AFS, DFS, NFS, RFS, SMTP, POP, DNS, Sun RPC, and NNTP.

5 One significant aspect of many applications of flow  
6 splicing is that the intermediate node may decide what  
7 node to establish either the first or second connection  
8 with based on several factors. For example the  
9 intermediate node may determine what node to establish  
10 the second connection with after parsing a request read  
11 from the first connection. The intermediate node might  
12 also determine what specific port or address within the  
13 first node to establish the first connection with based  
14 on the remote address associated with the second  
15 connection. The intermediate node might do this to  
16 direct requests from certain clients to certain  
17 servers. Other factors the intermediate node may  
18 consider include estimates of bandwidth and latency for  
19 the various flows, quality of service (QoS)  
20 requirements for the various flows, the number of  
21 network hops between any of the nodes, and  
22 configuration or state information stored at the  
23 intermediate node.

24 In addition to determining what address a connection  
25 should be established with, the intermediate node may  
26 determine whether to perform a splice at all based on  
27 several factors. For example an intermediate node may  
28 parse requests sent by a client to a server in order to  
29 estimate the amount of data expected to flow from the  
30 server to the client and splice only flows expected to  
31 handle a lot of data. The intermediate node might also  
32 splice only flows expected to support high bandwidth

1 and/or low latency. Finally the intermediate node might  
2 splice only those flows associated with a specific set  
3 of clients or servers.

4 We now describe a method that embodies the splice  
5 operation for the specific case where both flows joined  
6 by the splice are associated with TCP connections. It  
7 will be understood by those skilled in the art that the  
8 concept of transforming two flows, one inbound the  
9 other outbound at a given point, into a single flow  
10 from the source of the original inbound flow to the  
11 destination of the original outbound flow by suitably  
12 modifying packet headers can be applied to any packet  
13 based communication protocol.

14 Let us now consider TCP state. Every TCP end point  
15 pair has a set of associated state information. This  
16 information is referred to in the following discussion  
17 by the name TCB which stands for TCP Control Block.  
18 The specific way in which the state information is  
19 stored in a TCB is not important to the following  
20 discussion. The discussion does however refer to some  
21 specific elements of the TCB which are described as  
22 follows.

23 snd\_nxt - The sequence number of the next byte to be  
24 sent. One greater than the greatest sequence number  
25 sent so far.

26 snd\_una - The smallest sequence number associated  
27 with any byte that has been sent, but has not yet been  
28 acknowledged. Equal to the greatest acknowledgment  
29 value received.

30 rcv\_nxt - The sequence number of the next byte of  
31 data expected to arrive. One more than the greatest  
32 sequence number received.

1     rcv\_wnd - An offset from rcv\_nxt that identifies the  
2 largest sequence number a receiver is willing to  
3 receive.

4     Now we consider forward and reverse information.  
5 Figure 13 shows the structure of a TCP packet header.  
6 The information contained in a TCP packet may be  
7 divided into two categories. The first we call *forward*  
8 *outbound* information. This includes of the packet's  
9 data payload and associated sequence number 1303,  
10 checksum 1313, urgent pointer 1314, and any time stamp  
11 option value. The second category we call *reverse*  
12 *inbound* information. This includes the acknowledgment  
13 flag 1307 and field 1304, window size 1312, and any  
14 time stamp option echo value. The fields of the TCP  
15 header that contain reverse inbound information are  
16 shaded in Figure 13.

17     Certain fields of the TCP header and in the  
18 associated IP header identify the connection with which  
19 the segment is associated. These fields are the source  
20 and destination port number in the TCP header and the  
21 source and destination IP address in the IP header.  
22 Because these fields identify the connection, they are  
23 associated with both the forward outbound and reverse  
24 inbound information.

25     The forward information is intended for the end  
26 point pair to which data is sent on a TCP connection.  
27 The reverse information is intended for the end point  
28 pair from which data is received on the connection.  
29 Without flow splicing, these two end point pairs are  
30 identical. A splice, however, can break this equality.  
31 The ability to splice each flow associated with an end  
32 point pair independently relies on independent

1 processing of the forward outbound and reverse inbound  
2 information. The specific details of processing  
3 forward and reverse information in TCP packets is  
4 described in further detail below.

5 The next aspect to consider is splice states. A  
6 flow splice joins two flows into one thus eliminating a  
7 pair of flow end points. Several factors may make it  
8 impossible for the end points to be eliminated  
9 immediately when a splice is created. A splice  
10 therefore has an associated state value that influences  
11 its operation. We now discuss various factors that  
12 influence a splice's state and the operation of a  
13 splice in each state. A state diagram showing the  
14 various splice states and the logic for transitions  
15 between them is shown in Figure 14.

16 One consideration that can prevent spliced end  
17 points from being eliminated is the presence of data in  
18 send or receive buffers associated with the spliced  
19 flows. Each flow end point generally has an associated  
20 buffer. An inbound flow has a receive buffer that  
21 contains data that has been received but has not yet  
22 been read by the application. An outbound flow has a  
23 send buffer that contains data that has been written by  
24 the application but has not yet been successfully sent.  
25 Any data that resides in the send buffer when the  
26 splice is created is sent before packets can be  
27 forwarded via the splice. Similarly any data in the  
28 receive buffer is sent after any data in the send  
29 buffer and before any packets are forwarded.

30 If data resides in either the send or receive buffer  
31 when a splice is created, the splice starts in the  
32 PENDING 1402 state. Any data residing in the receive

1 buffer when the splice is created is appended to the  
2 contents, if any, of the send buffer. The receive  
3 buffer is then essentially eliminated as any data  
4 subsequently arriving at a splice in the PENDING 1402  
5 state is appended to the send buffer associated with  
6 the outbound flow instead of being appended to the  
7 receive buffer of the inbound flow as would be the case  
8 for data arriving on an unspliced flow.

9 Full protocol processing is performed for each end  
10 point associated with a splice in the PENDING 1402  
11 state. A PENDING 1402 splice behaves similarly to a  
12 program that reads data from the inbound flow as it  
13 arrives and writes it to the outbound flow. The splice  
14 however eliminates the need to invoke a separate  
15 program, possibly saving protection domain and context  
16 switches, and also eliminates any data copies between  
17 the protocol implementation and the program.

18 A splice remains in the PENDING 1402 state at least  
19 until its send buffer is empty. Additional data  
20 arriving at a PENDING 1402 splice may thwart attempts  
21 to empty the buffer. A PENDING 1402 splice may  
22 therefore limit the amount of data it receives by  
23 closing its *receive window*. The receive window is an  
24 item of reverse inbound information that informs a  
25 sender how much data a receiver is willing to receive.  
26 The TCP protocol forbids a receiver from decreasing its  
27 receive window by an amount greater than the amount of  
28 data it has received. In other words, once a receiver  
29 has indicated it is able to receive a given amount of  
30 data, it honors its commitment. However, if a receiver  
31 indicated it is capable of receiving X bytes of data

1 and subsequently receives Y bytes it is permitted to  
2 decrease its receive window to  $X - Y$ .

3 A PENDING TCP flow splice could also limit the  
4 amount of data it receives by withholding  
5 acknowledgments for data that it receives. The lack of  
6 acknowledgments will discourage the sender from sending  
7 additional data. Withholding acknowledgments may also  
8 cause the sender to retransmit data that has already  
9 been sent. For this reason it may be advantageous for  
10 a PENDING splice to limit the amount of data it  
11 receives by decreasing the send window rather than by  
12 withholding acknowledgments.

13 Regardless of the presence of buffered data, a  
14 splice is also prevented from proceeding beyond the  
15 PENDING 1402 state if the receive window advertised to  
16 the splice source (by the splice) is greater than the  
17 receive window advertised by the splice destination (to  
18 the splice). If this situation arises, the splice  
19 remains PENDING 1402 until it has received enough data  
20 to allow it to eliminate the gap between the receive  
21 window sizes.

22 Once the last byte of buffered data if any has been  
23 sent and the receive window has been decreased to the  
24 value advertised by the receiver if need be, the splice  
25 enters the FORWARDING 1403 state. In this state  
26 packets arriving at the splice are modified and  
27 forwarded. However, reverse inbound information  
28 associated with buffered data, as opposed to data  
29 simply forwarded through the splice, is handled in the  
30 same way as for an unspliced flow, rather than being  
31 forwarded to the sender. Once the splice is in the  
32 FORWARDING 1403 state, full protocol processing need

1 not be performed for the inbound flow. Processing of  
2 reverse inbound information, including performing  
3 retransmission still occurs, however, for the outbound  
4 flow.

5 Eventually the last reverse inbound information that  
6 refers to buffered data should be received at which  
7 point the splice enters the ESTABLISHED 1404 state. In  
8 this state both forward and reverse inbound information  
9 is processed (modified) and forwarded.

10 We now consider flow termination. When a node  
11 communicating via a splice terminates its flow, the  
12 termination can be handled by the splice in either of  
13 two ways. In one embodiment the splice propagates the  
14 termination, or *FIN* in TCP parlance just as it  
15 propagates ordinary data. This causes both flows  
16 joined by the splice to be shut down. In another  
17 embodiment the splice itself processes the *FIN*. This  
18 results in only the inbound flow at the splice being  
19 shut down and causes the outbound flow to revert to the  
20 UNSPLICED 1401 state. Having the splice propagate the  
21 *FIN* is desirable in situations where an intermediate  
22 node does not need to send any more data on the  
23 outbound flow after creating a splice. Having the  
24 splice process the *FIN* allows an intermediate node to  
25 either send additional data on the outbound flow, or  
26 resplice the flow.

27 Once a splice has processed a *FIN* and a  
28 corresponding *ACK*, it enters the WAITING 1405 state.  
29 This state is identical to the ESTABLISHED 1404 state  
30 except for the presence of a timer that causes the  
31 splice to cease to exist when it expires. The timer is  
32 set to  $2*MSL$ , twice the maximum segment lifetime, when



1 the splice enters the WAITING 1405 state and reset to  
2 2\*MSL whenever a packet is forwarded through the  
3 splice. Usually no packets are forwarded through the  
4 splice once it enters the WAITING 1405 state and the  
5 splice is eliminated 2\*MSL after the FIN and  
6 corresponding ACK are processed. The purpose of the  
7 WAITING state is to account for any retransmitted  
8 packets.

9 The 2\*MSL time-out is conservative in that the  
10 splice persists only until it can be determined that  
11 any retransmissions that might occur would have been  
12 seen at the splice. However, the retransmission  
13 time-out value used by a TCP sender is a function of  
14 both the estimated round trip time and its variance and  
15 it is difficult for the splice to accurately estimate  
16 these values as viewed by the sender.

17 Now consideration is given to sequence and  
18 acknowledgment number mapping. TCP is a stream  
19 oriented protocol. Although a sending client presents  
20 data to the protocol in discrete buffers and the  
21 protocol partitions transmitted data into packets,  
22 neither buffer nor packet boundaries are visible to the  
23 receiver. TCP presents data to a receiver as a  
24 continuous stream bounded only by a single pair of  
25 beginning and end points. To ensure data is presented  
26 to the receiver in exactly the same order in which it  
27 was sent, TCP assigns a 32-bit sequence number to each  
28 byte in the stream. The *initial sequence number*, that  
29 is the sequence number for the first byte to traverse a  
30 flow, is generally selected in a way that makes it  
31 difficult to predict and the sequence number is  
32 incremented by one for each successive byte.

1 Because different flows generally have different  
2 initial sequence numbers and because any amount of data  
3 can be sent on a flow before it is spliced, the  
4 sequence numbers for data flowing through a splice are  
5 mapped from the inbound to the outbound flow. For  
6 example, consider two flows shown in Figure 8,  $F_{AI}$  804  
7 which extends from node A 801 to node I 802, and  $F_{IB}$   
8 806 that extend from the node I 802 to node B 803.  
9 Assume the initial sequence number for  $F_{AI}$  804 (chosen  
10 by node A 801) is 1,000, and the initial sequence  
11 number for  $F_{IB}$  806 (chosen by the node I 802) is  
12 10,000. Further assume 200 bytes has arrived at I 802  
13 on  $F_{AI}$  804 and 300 bytes have been sent from I 802 on  
14  $F_{IB}$  806 before  $F_{AI}$  804 is spliced to  $F_{IB}$  806. When the  
15 splice is created, the sequence number of the next byte  
16 to be received on  $F_{AI}$  804 is 1,200 and the sequence  
17 number of the next byte to be sent on  $F_{IB}$  806 is  
18 10,300. Thus as data flows through the established  
19 splice the sequence number in each TCP packet is mapped  
20 from the inbound flow,  $F_{AI}$  804, to the outbound flow  $F_{IB}$   
21 806 by adding 9,100 ( $10,300 - 1,200$ ). Acknowledgment  
22 numbers flowing in the opposite direction are mapped by  
23 *subtracting* 9,100.

24 Creating a splice between two sequence-oriented  
25 flows establishes a correspondence between the sequence  
26 space of the first flow and the sequence space of the  
27 second flow. In the case of a splice between two TCP  
28 flows, the correspondence amounts to a fixed offset  
29 between the sequence number of a byte arriving at the  
30 intermediate node on the inbound flow and the sequence  
31 number of the corresponding byte sent by the  
32 intermediate node on the outbound flow.

1 The correspondence established when a splice is  
2 created can be calculated in a number of ways. For  
3 example, if when a splice is created, no data resides  
4 in either send or receive buffers associated with the  
5 flows to be spliced at the intermediate node, the  
6 offset can be calculated by subtracting the sequence  
7 number of the next byte expected from the source  
8 (rcv\_nxt) from the sequence number of the next byte to  
9 be sent to the destination (snd\_nxt). If either the  
10 receive buffer associated with the inbound flow or the  
11 send buffer associated with the outbound flow contains  
12 data, the sequence number of the next byte to be sent  
13 on the outbound flow is adjusted to account for the  
14 buffered data that will be sent before any data is  
15 forwarded through the splice. Alternately, the  
16 calculation of the offset can be delayed until both the  
17 send and receive buffer are empty.

18 We refer to the inbound and outbound flows at the  
19 splice even though the splice transforms the two flows  
20 into one. Strictly speaking, we are referring to the  
21 sequence number space associated with the former flow  
22 from the splice source to the intermediate node and the  
23 sequence number space associated with the former flow  
24 from the intermediate node to the splice destination.  
25 For brevity we simply refer to the inbound and outbound  
26 flows.

27 Processing for a TCP to TCP Splice in the  
28 established state is now considered. Figure 15 is a  
29 flow diagram that gives an overview of the logic for  
30 processing TCP segments associated with a flow splice  
31 in the ESTABLISHED 1404 state. The initial processing  
32 is the same regardless of whether or not the segment is

1 associated with a splice. First, in block 1502,  
2 several preliminary checks are performed to make  
3 certain the packet contains a valid TCP segment. The  
4 state information for the end point pair with which the  
5 segment is associated is then located. If no such  
6 state information is found, the segment is discarded  
7 and processing for the segment is complete. If the  
8 outbound flow of the end point pair on which the  
9 segment arrived is the destination of a splice the  
10 reverse inbound information contained in the segment is  
11 subjected to splice destination processing. This  
12 processing is described in further detail below and is  
13 depicted in Figure 16.

14 If a segment is subject to splice destination  
15 processing and contains neither data nor a FIN, then  
16 processing for the segment is complete. If a segment  
17 does contain either data or a FIN or the segment was  
18 not subject to splice destination processing the  
19 inbound flow of the end point pair on which the segment  
20 arrived is checked in decision block 1506 to see if it  
21 is a splice source. If the flow is not a splice  
22 source, the segment is subjected to the remainder of  
23 ordinary TCP input processing. If the inbound flow is  
24 a splice source, the corresponding outbound flow is  
25 checked in decision block 1507 to determine if it is  
26 not the destination of a splice and the segment is  
27 checked to see if it contains new reverse inbound  
28 information. If these two conditions are true, a copy  
29 of the reverse inbound information in the segment is  
30 made in block 1508 so it can be subjected to the  
31 remainder of the ordinary TCP input processing. A  
32 segment is considered to contain new reverse inbound

1 information if it contains an acknowledgment number  
 2 greater than the largest acknowledgment number  
 3 previously seen on this flow, or if it contains a  
 4 window update. The segment is next subjected to splice  
 5 source processing which is described in further detail  
 6 below and depicted in Figure 17. If a copy of the  
 7 segment's reverse inbound information was made, that  
 8 information is subjected to the remainder of ordinary  
 9 TCP input processing in block 1511. This 1512  
 10 completes processing for a TCP segment associated with  
 11 a splice.

12 Figure 16 is a flow diagram that depicts splice  
 13 destination processing. Processing begins in block  
 14 1601 and in 1602 the acknowledgment number in the  
 15 segment is mapped from the destination flow to the  
 16 source flow, and the window value in the segment is  
 17 unscaled and limited to the maximum value that can be  
 18 communicated to the splice source. The limiting factor  
 19 is required in cases where the node on which the splice  
 20 is performed has negotiated a larger window scale value  
 21 with the splice destination than with the splice  
 22 source. The mapped acknowledgment number is then  
 23 checked to determine if it is greater than the greatest  
 24 acknowledgment number sent from the splice to the  
 25 splice source (stored in the rcv\_nxt field of the  
 26 source TCB) 1603. If so the mapped acknowledgment  
 27 number is recorded 1604 in the rcv\_nxt field of the  
 28 source TCB and processing continues with formulating a  
 29 TCP segment containing reverse inbound information to  
 30 be sent to the splice source in block 1606. Block 1606  
 31 includes:

32 1. Allocate packet for TCP segment,

1     2. Fill in fields of IP header needed for TCP  
2 checksum (packet length, protocol, source, destination  
3 IP address),  
4     3. Fill in fields of TCP header (source, destination  
5 port, TCP header length, sequence and acknowledgment  
6 numbers, flags).  
7     If the mapped acknowledgment value is not greater  
8 than the rcv\_nxt field of the source TCB, the segment  
9 is checked to determine if it contains any data or a  
10 FIN 1605. If the segment contains either data or a FIN  
11 processing for the segment is complete. If not,  
12 processing continues with formulating a TCP segment  
13 containing reverse inbound information in block 1606.  
14     Formulating the TCP segment begins with allocating a  
15 new packet and filling in the fields of the IP pseudo  
16 header needed for the IP checksum, specifically the  
17 packet length, protocol, and source and destination IP  
18 address. The IP addresses are copied from the source  
19 TCB. Next the fields of the TCP header are filled in.  
20 These are the source and destination port numbers,  
21 sequence and acknowledgment numbers, TCP header length  
22 and flags. The port numbers are copied from the source  
23 TCB. The only flag set is ACK. The sequence number is  
24 copied from the snd\_nxt field of the source TCB. The  
25 acknowledgment field is set to the mapped  
26 acknowledgment value.  
27     A determination is made to see if the segment  
28 contains a window update 1607. If it does, the value  
29 for the receive window, and sequence and acknowledgment  
30 numbers are recorded in the destination TCB 1608. Then  
31 block 1609 is performed and the receive window field of  
32 the TCP header is copied from the destination TCB. The  
33 TCP checksum is calculated and filled in. The total  
34 length, type of service and time to live fields of the

1 IP header are filled in. Finally the packet is passed  
2 to the IP output routine and the process ends 1610.

3 In the embodiment described new reverse inbound  
4 information is propagated to the splice source  
5 immediately. The sending of reverse inbound  
6 information could also be delayed for a short period of  
7 time in the hope that it could be sent in segments  
8 containing forward data for the source instead of  
9 sending a packet containing only reverse inbound  
10 information.

11 Figure 17 is a flow diagram that depicts the steps  
12 of an example of splice source processing. The  
13 majority of processing is performed in block 1702 and  
14 includes modifying information in the TCP and IP packet  
15 headers. Step 1702 includes:

- 16 1. Modify source and destination address in IP
- 17 header
- 18 2. Modify source and destination port in TCP header
- 19 3. Map sequence number from source to destination
- 20 flow
- 21 4. Set acknowledgment field from rcv\_nxt field of
- 22 destination TCB
- 23 5. Set window field from rcv\_wnd field of
- 24 destination TCB
- 25 6. Set length field in IP pseudo header
- 26 7. Calculate and fill in TCP checksum
- 27 8. Fill in IP total length, type of service and time
- 28 to live fields

29 The source and destination IP addresses and port  
30 numbers are modified by copying the appropriate values  
31 from the destination TCB. The sequence number is  
32 mapped from the source flow to the destination flow.

1 The acknowledgment number and receive window fields are  
2 set by copying the values from the rcv\_nxt and rcv\_wnd  
3 fields of the destination TCB respectively. The length  
4 field in the IP pseudo header is set. The TCP checksum  
5 is calculated and filled in. The total length, type of  
6 service, and time to live fields of the IP header are  
7 filled in.

8 If the segment's FIN flag is set TCP state  
9 processing is performed for both the source and  
10 destination end point pairs. The processing for the  
11 source pair is the same as that performed when a FIN is  
12 received on an unspliced flow. The processing for the  
13 destination pair is the same as what takes place when a  
14 FIN is sent on an unspliced flow. Finally, the segment  
15 is passed to the IP output routine.

16 As described above, the preferred embodiment of TCP  
17 flow splicing recomputes the TCP checksum for each  
18 packet forwarded through a splice after the packet  
19 header as been modified. Techniques in common practice  
20 allow the new TCP checksum to be calculated by  
21 referring to only the original TCP checksum value and  
22 the changes made to the packet. This can improve  
23 performance compared to computing the checksum from  
24 scratch because it eliminates the need to reference all  
25 the data in the packet.

26 Preliminary checks are performed on the TCP segment  
27 in block 1502 of Figure 15. These checks may or may  
28 not include TCP checksum calculation. Performing the  
29 checksum allows corrupt packets to be detected and  
30 eliminated as soon as possible which may save network  
31 bandwidth. Processing requirements are reduced,  
32 however, if the checksum calculation is eliminated from



1 the splice. Elimination of the checksum is feasible as  
2 corrupt packets will still be detected by normal TCP  
3 processing at the destination node.

4 The flow splice may be defined in terms of packet  
5 sequences. Figure 18 depicts an example of a sequence  
6 of packets associated with the scenario depicted in  
7 Figure 8. In this scenario, each of two nodes A 801  
8 and B 803 has established a connection with an  
9 intermediate node I 802. In addition to the elements  
10 explicitly depicted in Figure 8, the scenario  
11 associated with Figure 18 includes a program on node I  
12 802 that continuously reads data from end point pair  
13  $EP_{I1}$  811 and immediately writes any such data on end  
14 point pair  $EP_{I2}$  812. This corresponds to reading data  
15 from flow  $F_{A1}$  804 and writing it to flow  $F_{I2}$  806.

16 The packet flow depicted in Figure 18 results from  
17 node A 801 sending 100 bytes of data on end point pair  
18  $EP_{A1}$  810. These 100 bytes are contained in a single  
19 packet 1801. Subsequent to receiving this packet 1801,  
20 node I 802 sends an acknowledgment packet 1802 to node  
21 A 801. The acknowledgment packet 1802 indicates to  
22 node A 801 that the 100 bytes contained in the first  
23 packet 1801 have successfully arrived at the  
24 destination of the flow on which they were sent, that  
25 is the destination of flow  $F_{A1}$  804 which is node I 802.

26 Also subsequent to the arrival at node I 802 of  
27 packet 1801, node I 802 sends the 100 bytes of data  
28 received from node A 801 on end point pair  $EP_{I2}$  812.  
29 Again the data happens to be conveyed in a single  
30 packet 1803. Subsequent to the arrival at node B 803  
31 of this packet 1803, node B 803 sends an acknowledgment  
32 packet 1804 to node I 802. This acknowledgment packet

1 1804 indicates to node I 802 that the 100 bytes  
2 contained in packet 1803 have successfully arrived at  
3 the destination of the flow on which they were sent,  
4 that is the destination of flow  $F_{IB}$  806 which is node B  
5 803.

6 Although Figure 18 depicts a specific strict  
7 temporal ordering of the transmission and reception of  
8 the four packets 1801, 1802, 1803 and 1804, certain  
9 other orderings are consistent with the depicted  
10 scenario. The only requirements on the temporal  
11 ordering are that each of packets 1802 and 1803 is sent  
12 from node I 802 *after* packet 1801 arrives at node I  
13 802. Furthermore packet 1804 is sent from node B 803  
14 *after* packet 1803 arrives at node B 803. Of course a  
15 packet can not arrive at its destination prior to being  
16 sent from its source.

17 The conclusive indication that the packet flow in  
18 Figure 18 is associated with two flows that are not  
19 joined in a splice is that node I 802 sends to node A  
20 801 an acknowledgment 1802 for the 100 bytes it  
21 received from node A 801 in packet 1801 *before* node I  
22 802 receives from node B 803 an acknowledgment for the  
23 corresponding 100 bytes sent to node B 803 in packet  
24 1803. This order indicates that the acknowledgment  
25 1802 sent by node I 802 is generated by normal TCP  
26 processing and not a flow splice at node I 802.

27 Figure 19 depicts a sequence of packets associated  
28 with the scenario depicted in Figure 9. This scenario  
29 is identical to the scenario shown in Figure 8 except  
30 that in Figure 9 flow  $F_{AI}$  804 inbound to node I 802 has  
31 been spliced to flow  $F_{IB}$  806 outbound from node I 802

1 thus transforming the two flows into a single flow  $F_{AB}$   
2 904.

3 Like the packet sequence depicted in figure 18 the  
4 flow shown in 19 results from node A 801 sending 100  
5 bytes of data on end point pair  $EP_{A1}$  907. Again, these  
6 100 bytes are contained in a single packet 1901.  
7 Subsequent to receiving this packet 1901, node I 802  
8 forwards the packet to node B 803 after modifying the  
9 packet headers to make it appear to node B 803 as if  
10 the packet had been sent from node I 802 on flow  $F_{IB}$   
11 806. Unlike the packet exchange shown in Figure 18 the  
12 exchange in Figure 19 does not include an  
13 acknowledgment from node I 802 to node A 801 in  
14 response to data packet 1901 from node A 801 arriving  
15 at node I 802.

16 Like Figure 18, Figure 19 shows node B 803  
17 generating an acknowledgment packet 1903 in response to  
18 the arrival of a packet 1902 from node I 802. As  
19 described earlier the splice mechanism creates packets  
20 containing data forwarded from node A 801 through node  
21 I 802 to node B 803 on flow  $F_{AB}$  904 that are acceptable  
22 to node B 803 as packets that originated at node I 802  
23 and were sent to node B 803 via flow  $F_{IB}$  806. When the  
24 acknowledgment packet 1903 arrives at node I 802, node  
25 I 802 modifies the packet to make it appear to node A  
26 801 that the packet originated at node I 802 and is  
27 associated with flow  $F_{AI}$  804. Node I 802 then forwards  
28 the modified packet 1904 to node A 801.

29 The temporal order of the packets in Figure 19 is  
30 the only one consistent with a spliced flow. That is,  
31 for a spliced flow packet 1904 is not sent from node I  
32 802 before packet 1903 arrives at node I 802, and

1 packet 1902 is not sent from node I 802 before packet  
2 1901 arrives at node I 802. Packet 1903 is not sent  
3 from node B 803 before packet 1902 arrives at node B  
4 803 simply to conform with the TCP protocol.

5 By itself this temporal ordering is not sufficient  
6 proof of a flow splice as it is also possible for this  
7 order to result without a splice. The presence of a  
8 flow splice is conclusively indicated by *causality*  
9 between packets received by and sent by the  
10 intermediate node I 802 on which the splice is  
11 performed. Specifically, in the scenario with a flow  
12 splice depicted in Figure 19, the sending of  
13 acknowledgment packet 1904 by node I 802 is caused by  
14 the arrival of acknowledgment packet 1903 from node B  
15 803 and not by the arrival of data packet 1901 from  
16 node A 801.

17 This causality is refuted if node I 802 sends an  
18 acknowledgment to node A 801 for data packet 1901  
19 before an acknowledgment is received at node I 802 for  
20 the corresponding data packet 1902 sent from node I 802  
21 to node B 803. The causality indicative of a flow  
22 splice is confirmed if node I 802 fails to send an  
23 acknowledgment to node A 801 for packet 1901 if it does  
24 not receive the acknowledgment packet 1903 from node B  
25 803 for the corresponding packet 1902.

26 The causality indicative of a flow splice can also  
27 be confirmed a different way. In the scenario without  
28 a flow splice depicted in Figure 18 the ACK number in  
29 packet 1802 is set by node I 802 to one more than the  
30 greatest sequence number I 802 has received from A 801  
31 on flow  $F_{AI}$  804 as mandated by the TCP protocol. In  
32 the case depicted in Figure 19, node I 802 received

1 from A 801 100 bytes starting at sequence number 1000  
2 in packet 1801. Packet 1801 contains sequence numbers  
3 1000 through 1099. Node I 802 therefore sets the ACK  
4 number in packet 1802 to 1100.

5 In the scenario with a splice depicted in Figure 19  
6 the ACK number in packet 1904 is set by node I 802 to  
7 the ACK number in the corresponding packet 1903 minus  
8 the *sequence number delta* value for the splice  
9 following the algorithm for splice processing. The  
10 sequence number delta value for a splice is defined as  
11 the sequence number of a byte sent by the intermediate  
12 node to the splice destination minus the sequence  
13 number of the corresponding byte when received from the  
14 splice source. For example the first of the 100 bytes  
15 in packet 1902, sent from node I 802 to node B 803, has  
16 sequence number 2000. The corresponding byte was sent  
17 from node A 801 to node I 802 in packet 1901 with  
18 sequence number 1000. This the sequence number delta  
19 for this splice is therefore 2000 minus 1000 or 1000.  
20 Thus when node I 802 receives acknowledgment packet  
21 1903 from node B 803 it subtracts 1000 from the ACK  
22 number 2100 and sets the ACK number in the forwarded  
23 packet 1904 to 1100.

24 Although the ACK number in each of packets 1802 and  
25 1904 are set according to distinct algorithms, both  
26 algorithms produced the same value 1100. The flow  
27 splice processing is, in fact, required to produce the  
28 same ACK numbers as produced by normal TCP processing  
29 if it is to remain transparent to the splice source and  
30 destination. Although under normal operation the two  
31 algorithms produce the same ACK number, the specific  
32 algorithm in use at a particular intermediate node can

1 be determined by modifying the acknowledgment packets  
2 sent to the intermediate node.

3 For example to determine if a flow splice has been  
4 created at node I 802, we simply add some amount to the  
5 ACK number in the acknowledgment packet 1903 sent by  
6 node B 803. If a flow splice is not present this  
7 modification will have no effect on the ACK number in  
8 the acknowledgment packet 1904 sent by the intermediate  
9 node I 802. If a flow splice is present, however, the  
10 ACK number in the acknowledgment packet 1904 sent by  
11 the intermediate node I 802 will reflect the  
12 modification to the ACK number in the acknowledgment  
13 packet sent by the destination node B 803.

14 If 1000 is added to the ACK number in the  
15 acknowledgment packet 1903 sent by the destination node  
16 B 803, the presence of a flow splice is indicated if  
17 the ACK number in the corresponding acknowledgment  
18 packet 1904 sent by the intermediate node I 802 is also  
19 1000 more than would be expected according to the TCP  
20 algorithm. It is certain the TCP algorithm did not  
21 generate the acknowledgment packet sent by the  
22 intermediate node I 802 if the ACK number in the packet  
23 corresponds to data not yet received by the  
24 intermediate node I 802.

25 It is noted that this invention may be used for many  
26 applications. Although the description is made for  
27 particular arrangements and applications, the intent  
28 and concept of the invention is suitable and applicable  
29 to other arrangements and applications. It will be  
30 clear to those skilled in the art that other  
31 modifications to the disclosed embodiments can be

1 effected without departing from the spirit and scope of  
2 the invention.  
3 .

1 Claims

2 Having thus described our invention, what we claim  
3 as new and desire to secure by Letters Patent is as  
4 follows:

5 1. A method for processing network packets at an  
6 intermediate node, the method comprising:

7 forming a first connection between a first node and  
8 the intermediate node this first connection including:

9 a first flow originating at a first source  
10 flow end point on the first node and terminating at  
11 a first destination flow end point on the  
12 intermediate node,, wherein processing at the first  
13 node associated with the first source flow end point  
14 conforms to a first protocol, and

15 a second flow originating at a second source  
16 flow end point on the intermediate node and  
17 terminating at a second destination flow end point  
18 on the first node, wherein processing at the first  
19 node associated with the second destination flow end  
20 point conforms to a second protocol;

21 forming a second connection between a second node  
22 and the intermediate node this second connection  
23 including:

24 a third flow originating at a third source  
25 flow end point on the intermediate node and



1 terminating at a third destination flow end point on  
2 the second node, wherein processing at the second  
3 node associated with the third destination flow end  
4 point conforms to a third protocol, and

5 a fourth flow originating at a fourth source  
6 flow end point on the second node and terminating at  
7 a fourth destination flow end point on the  
8 intermediate node, wherein processing at the second  
9 node associated with the fourth source flow end  
10 point conforms to a fourth protocol,

11 such that a given flow originates at a source flow  
12 end point on a source node and terminates at a  
13 destination flow end point on a destination node and  
14 data written to the source flow end point of a given  
15 flow can subsequently be read from the destination  
16 flow end point of the given flow without traversing  
17 any intervening flow end points, and

18 splicing the first flow and third flow to form a  
19 first composite flow originating at the first source  
20 flow end point on the first node and terminating at the  
21 third destination flow end point on the second node in  
22 a manner whereby the second flow and the fourth flow  
23 remain unchanged.

24 2. A method as recited in claim 1, wherein the  
25 step of splicing the first flow and third flow to form  
26 the first composite flow allows

1     processing at the first node associated with the  
2 first source flow end point to remain unmodified and  
3 continue to conform to the first protocol,  
4     processing at the first node associated with the  
5 second destination flow end point to remain unmodified  
6 and continue to conform to the second protocol,  
7     processing at the second node associated with the  
8 third destination flow end point to remain unmodified  
9 and continue to conform to the third protocol, and  
10    processing at the second node associated with the  
11 fourth source flow end point to remain unmodified and  
12 continue to conform to the fourth protocol.

13    3. A method recited in claim 1, wherein the first  
14 node and the second node are the same node.

15    4. A method recited in claim 1, wherein the first  
16 node and intermediate node are the same node.

17    5. A method recited in claim 1, wherein the second  
18 node and the intermediate node are the same node.

19    6. A method recited in claim 1, wherein the first  
20 node, the second node, and the intermediate node are  
21 all the same node.

22    7. A method recited in claim 1, wherein the first  
23 protocol and the second protocol are the same protocol.

24    8. A method recited in claim 1, wherein the third  
25 protocol and the fourth protocol are the same protocol.

1     9. A method recited in claim 1, wherein the first  
2 protocol, the second protocol, the third protocol, and  
3 the fourth protocol are all the same protocol.

4     10. A method recited in claim 1, wherein the step of  
5 splicing the first flow and third flow to form the  
6 first composite flow includes:

7     identifying a first set of packets received from the  
8 first node including all packets containing information  
9 pertaining to the first source flow end point and all  
10 packets containing information pertaining to the second  
11 destination flow end point and performing the following  
12 four steps (a), (b), (c) and (d) on each packet in  
13 this first set of packets;

14     (a) processing any information in the packet  
15 pertaining to the second flow according to the  
16 second protocol;

17     (b) replacing any information in the packet  
18 pertaining to the second flow with the corresponding  
19 information pertaining to the fourth flow;

20     (c) modifying the packet so that any information in  
21 the packet pertaining to the flow from the first  
22 source flow end point will appear to the second node  
23 as pertaining to the flow to the third destination  
24 flow end point thus establishing a correspondence  
25 between data received by the intermediate node from  
26 the first source flow end point and data sent by the

1 intermediate node to the third destination flow end  
2 point, and

3 (d) sending each packet so processed to the second  
4 node;

5 identifying a second set of packets received from  
6 the second node including all packets containing  
7 information pertaining to the third destination flow  
8 end point and all packets containing information  
9 pertaining to the fourth source flow end point and  
10 performing the following steps (d), (e), and (f), on  
11 each packet in this second set of packets;

12 (d) identifying any information in the packet  
13 pertaining to the fourth flow and processing such  
14 information according to the fourth protocol;

15 (e) modifying any information in the packet  
16 pertaining to the flow to the third destination flow  
17 end point so the information instead pertains to the  
18 flow from the first source flow end point according to  
19 the correspondence between data received by the  
20 intermediate node from the first source flow end point  
21 and data sent by the intermediate node to the third  
22 destination flow end point established in step (b);

23 (f) sending to the first node zero or more packets  
24 containing any information resulting from step (e).

1 11. A method recited in claim 10, wherein step (c)  
2 includes determining if the packet to be sent should be  
3 fragmented and fragmenting it if so.

4 12. A method recited in claim 10, further comprising  
5 determining if each packet in the first set of packets  
6 contains data written to the first source flow end  
7 point and nullifying steps (b), (c) and (d) if not.

8 13. A method recited in claim 10, further comprising  
9 determining if each packet in the second set of packets  
10 contains data written to the fourth source flow end  
11 point, determining if each packet in the second set of  
12 packets contains no information pertaining to the flow  
13 to the third destination flow end point that has not  
14 already been processed by a previous application of  
15 step (e), and nullifying steps (e) and (f) for a given  
16 packet if both determinations are true for the given  
17 packet.

18 14. A method recited in claim 1, wherein the first  
19 protocol and the third protocol each associate a  
20 sequence number with each byte, packet or other unit of  
21 data sent across a flow further comprising the step of  
22 maintaining a one to one mapping between sequence  
23 numbers associated by the first protocol with each  
24 byte, packet, or other unit of data received by the  
25 intermediate node from the first source flow end point  
26 and sequence numbers associated by the second protocol  
27 with each byte, packet, or other unit of data sent by  
28 the intermediate node to the third destination flow end  
29 point.

1 15. A method recited in claim 1, further comprising  
2 the step of sending data stored in buffers associated  
3 with the third source flow end point to the third  
4 destination flow end point according to the third  
5 protocol.

6 16. A method recited in claim 1, further comprising  
7 the step of sending data stored in buffers associated  
8 with the first destination flow end point to the third  
9 destination flow end point according to the third  
10 protocol.

11 17. A method recited in claim 1, further comprising  
12 the step of determining if the first node has shut down  
13 the flow originating at the first source flow end point  
14 and shutting down the flow terminating at the third  
15 destination flow end point if this determination is  
16 true.

17 18. A method recited in claim 1, further comprising  
18 determining if the first node has shut down the flow  
19 originating at the first source flow end point and  
20 eliminating the first composite flow and recreating the  
21 third flow if this determination is true.

22 19. A method recited in claim 1, further comprising  
23 the step of making a copy of data received from the  
24 first node by the intermediate node such that the copy  
25 may subsequently be read at the intermediate node.

1     20. A method recited in claim 1, further comprising  
2 the steps of:

3     reading an amount of data from the fourth  
4 destination flow end point;

5     storing the amount of data in a format satisfying a  
6 need from the group of needs consisting of:

7             reducing the number of bits required to  
8 store the data,

9             encrypting the data, reducing the number or  
10 amount of resources required to transmit the data,

11            reducing the number or amount of resources  
12 required to display the data, and

13            any combination of these needs; and

14     sending the stored data to the first node.

15     21. An apparatus comprising:

16     a first node, a second node, an intermediate node;

17     a first processor that forms a first connection  
18 between a first node and the intermediate node the  
19 first connection including:

20             a first flow originating at a first source  
21 flow end point on the first node and terminating at  
22 a first destination flow end point on the  
23 intermediate node, wherein processing at the first  
24 node associated with the first source flow end point  
25 conforms to a first protocol, and

1           a second flow originating at a second source  
2   flow end point on the intermediate node and  
3   terminating at a second destination flow end point  
4   on the first node, wherein processing at the first  
5   node associated with the second destination flow end  
6   point conforms to a second protocol;

7   a second processor that forms a second connection  
8   between a second node and the intermediate node the  
9   second connection including:

10           a third flow originating at a third source  
11   flow end point on the intermediate node and  
12   terminating at a third destination flow end point on  
13   the second node, wherein processing at the second  
14   node associated with the third destination flow end  
15   point conforms to a third protocol, and

16           a fourth flow originating at a fourth source  
17   flow end point on the second node and terminating at  
18   a fourth destination flow end point on the  
19   intermediate node, wherein processing at the second  
20   node associated with the fourth source flow end  
21   point conforms to a fourth protocol,

22   such that a given flow originates at a source flow  
23   end point on a source node and terminates at a  
24   destination flow end point on a destination node and  
25   data written to the source flow end point of a given  
26   flow can subsequently be read from the destination



1 flow end point of the given flow without traversing  
2 any intervening flow end points, and

3 a third processor that splices the first flow and  
4 third flow to form a first composite flow originating  
5 at the first source flow end point on the first node  
6 and terminating at the third destination flow end point  
7 on the second node in a manner whereby the second flow  
8 and the fourth flow remain unchanged.

9 22. A method recited in claim 1, further comprising:

10 forming a third connection between a third node and  
11 the intermediate node the third connection including:

12 a fifth flow originating at a fifth source flow end  
13 point on the intermediate node and terminating at a  
14 fifth destination flow end point on the third node,  
15 wherein processing at the third node associated with  
16 the fifth destination flow end point conforms to a  
17 fifth protocol, and

18 a sixth flow originating at a sixth source flow end  
19 point on the third node and terminating at a sixth  
20 destination flow end point on the intermediate node,  
21 wherein processing at the sixth node associated with  
22 the fourth source flow end point conforms to a sixth  
23 protocol,

24 splicing the sixth flow and second flow to form a  
25 third composite flow originating at the sixth source  
26 flow end point on the third node and terminating at the

1 second destination flow end point on the first node in  
2 a manner whereby the fourth flow and the fifth flow  
3 remain unchanged.

4 23. A method recited in claim 1, further comprising  
5 the step of reading data from the first destination  
6 flow end point prior to the step of splicing.

7 24. A method recited in claim 1, further comprising  
8 the step of writing data to the third source flow end  
9 point prior to the step of splicing.

10 25. A method recited in claim 1, further comprising  
11 the step of determining if the step of splicing should  
12 be performed based on a criterion selected from the  
13 group of criteria consisting of:  
14 data read from the first destination flow end point,  
15 data read from the fourth destination flow end  
16 point,  
17 data written to the second source flow end point,  
18 data written to the third source flow end point,  
19 an address associated with the first node,  
20 an address associated with the second node,  
21 an address associated with the intermediate node,  
22 an address associated with the first flow,  
23 an address associated with the second flow,  
24 an address associated with the third flow,  
25 an address associated with the fourth flow,  
26 an estimate of the available bandwidth along the  
27 first flow,  
28 an estimate of the available bandwidth along the  
29 second flow,

1 an estimate of the available bandwidth along the  
2 third flow,  
3 an estimate of the available bandwidth along the  
4 fourth flow,  
5 an estimate of the end to end latency associated  
6 with the first flow,  
7 an estimate of the end to end latency associated  
8 with the second flow,  
9 an estimate of the end to end latency associated  
10 with the third flow,  
11 an estimate of the end to end latency associated  
12 with the fourth flow,  
13 a receive window size advertised by the first node,  
14 a receive window size advertised by the second node,  
15 a receive window size advertised by the intermediate  
16 node,  
17 a number of network hops between the first node and  
18 the intermediate node,  
19 a number of network hops between the intermediate  
20 node and the second node,  
21 a number of network hops between the first node and  
22 the second node,  
23 a protocol associated with the first flow,  
24 a protocol associated with the second flow,  
25 a protocol associated with the third flow,  
26 a protocol associated with the fourth flow,  
27 an estimate of the amount of data that will be  
28 written to the first source flow end point,  
29 an estimate of the amount of data that will be  
30 written to the second source flow end point,  
31 an estimate of the amount of data that will be  
32 written to the third source flow end point,

1 an estimate of the amount of data that will be  
2 written to the fourth source flow end point,  
3 configuration information stored at or accessible  
4 from the intermediate node,  
5 information received from the first node pertaining  
6 to quality of service,  
7 information received from the second node pertaining  
8 to quality of service , and  
9 any combination of these criteria.

10 26. A method recited in claim 1, further comprising  
11 the step of determining the network addresses with  
12 which the first flow, second flow, third flow, and  
13 fourth flow are established based on a criterion  
14 selected from the group of criteria consisting of:  
15 data read from the first destination flow end point,  
16 data read from the fourth destination flow end  
17 point,  
18 data written to the second source flow end point,  
19 data written to the third source flow end point,  
20 a network address associated with a potential or  
21 actual first node,  
22 a network address associated with a potential or  
23 actual second node,  
24 a network address associated with the intermediate  
25 node,  
26 an address associated with the first flow,  
27 an address associated with the second flow,  
28 an address associated with the third flow,  
29 an address associated with the fourth flow,  
30 an estimate or prediction of the available bandwidth  
31 along the first flow

1 an estimate or prediction of the available bandwidth  
2 along the second flow,  
3 an estimate or prediction of the available bandwidth  
4 along the third flow  
5 an estimate or prediction of the available bandwidth  
6 along the fourth flow,  
7 an estimate or prediction of the end to end latency  
8 along the first flow,  
9 an estimate or prediction of the end to end latency  
10 along the second flow,  
11 an estimate or prediction of the end to end latency  
12 along the third flow,  
13 an estimate or predication of the end to end latency  
14 along the fourth flow,  
15 a receive window size advertised by the first node,  
16 a receive window size advertised by the second node,  
17 a receive window size advertised by the intermediate  
18 node,  
19 a number of network hops between the first node and  
20 the intermediate node,  
21 a number of network hops between the intermediate  
22 node and the second node,  
23 a number of hops between the first node and the  
24 second node,  
25 a protocol associated with the first flow,  
26 A protocol associated with the second flow,  
27 an estimate of the amount of data that will be  
28 written to the first source flow end point,  
29 an estimate of the amount of data that will be  
30 written to the second source flow end point,  
31 an estimate of the amount of data that will be  
32 written to the third source flow end point,

1 an estimate of the amount of data that will be  
2 written to the fourth source flow end point,  
3 configuration information stored at or accessible  
4 from the intermediate node,  
5 information received from the first node pertaining  
6 to quality of service,  
7 information received from the second node pertaining  
8 to quality of service, and  
9 any combination of these criteria.

10 27. A method recited in claim 10, wherein each of  
11 the first protocol, the second protocol, the third  
12 protocol and the fourth protocol are Transmission  
13 Control Protocol (TCP) in conjunction with either  
14 version 4 or version 6 of Internet Protocol (IP) and  
15 wherein step (c) of claim 1, includes the following  
16 steps (a), (b), (c), (d), and (e):

17 (a) setting the source IP address in the IP header  
18 to the local IP address associated with the third  
19 source flow end point;  
20 (b) setting the destination IP address in the IP  
21 header to the remote IP address associated with the  
22 third source flow end point;  
23 (c) setting the source port number in the TCP header  
24 to the local port number associated with the third  
25 source flow end point;  
26 (d) setting the destination port number in the TCP  
27 header to the remote port number associated with the  
28 third source flow end point, and  
29 (e) modifying the sequence (SEQ) number in the TCP  
30 header;

1 and wherein step (b) of claim 10, includes the  
2 following steps (f), (g), (h), and (i):

3 (f) replacing the acknowledgment (ACK) number in the  
4 TCP header;

5 (g) replacing the window value in the TCP header;

6 (h) modifying or recalculating the TCP checksum in  
7 the TCP header, and

8 (i) modifying or recalculating the IP checksum in  
9 the IP header;

10 and wherein step (e) of claim 1, includes the  
11 following steps (j), (k), (l), (m), (n), (o), (p) and  
12 (q):

13 (j) setting the source IP address in the IP header  
14 to the local IP address associated with the first  
15 destination flow end point;

16 (k) setting the destination IP address in the IP  
17 header to the remote IP address associated with the  
18 first destination flow end point;

19 (l) setting the source port number in the TCP header  
20 to the local port number associated with the first  
21 destination flow end point;

22 (m) setting the destination port number in the TCP  
23 header to the remote port number associated with the  
24 first destination flow end point;

25 (n) replacing the sequence (SEQ) number in the TCP  
26 header;

27 (o) modifying the acknowledgment (ACK) number in the  
28 TCP header;

1 (p) modifying the TCP checksum in the TCP header,  
2 and  
3 (q) modifying the IP checksum in the IP header.

4 28. A method recited in claim 27, wherein any of the  
5 checksum calculations performed in steps (h), (i), (p),  
6 and (q) is calculated by adjusting the original  
7 checksum to account for changes made to the packet at  
8 the intermediate node.

9 29. A method recited in claim 27, wherein the first  
10 connection extends between a TCP server on the first  
11 node and a SOCKS server on the intermediate node and  
12 the second connection extends between the SOCKS server  
13 on the intermediate node and a SOCKS client on the  
14 second node further comprising parsing data received  
15 from either the SOCKS client or the TCP server.

16 30. A method recited in claim 27, wherein the first  
17 connection extends between a SOCKS client on the first  
18 node and a SOCKS server on the intermediate node and  
19 the second connection extends between the SOCKS server  
20 on the intermediate node and a TCP server on the second  
21 node further comprising parsing data received from  
22 either the SOCKS client or the TCP server.

23 31. A method recited in claim 27, wherein the first  
24 connection extends between a Sun remote procedure call  
25 (Sun RPC) server on the first node and a proxy on the  
26 intermediate node and the second connection extends  
27 between the proxy on the intermediate node and a Sun  
28 RPC client on the second node further comprising



1 parsing data received from either the Sun RPC client or  
2 the Sun RPC server.

3 32. A method recited in claim 27, wherein the first  
4 connection extends between a Sun remote procedure call  
5 (Sun RPC) client on the first node and a proxy on the  
6 intermediate node and the second connection extends  
7 between the proxy on the intermediate node and a Sun  
8 RPC server on the second node further comprising  
9 parsing data received from either the Sun RPC client or  
10 the Sun RPC server.

11 33. A method recited in claim 27, wherein the first  
12 connection extends between a hypertext transfer  
13 protocol (HTTP) server on the first node and a proxy on  
14 the intermediate node and the second connection extends  
15 between the proxy on the intermediate node and an HTTP  
16 client on the second node further comprising parsing  
17 data received from either the HTTP client or the HTTP  
18 server.

19 34. A method recited in claim 27, wherein the first  
20 connection extends between a hypertext transfer  
21 protocol (HTTP) client on the first node and a proxy on  
22 the intermediate node and the second connection extends  
23 between the proxy on the intermediate node and an HTTP  
24 server on the second node further comprising parsing  
25 data received from either the HTTP client or the HTTP  
26 server.

27 35. A method recited in claim 27, wherein the first  
28 connection extends between a file transfer protocol

1 (FTP) server on the first node and a proxy on the  
2 intermediate node and the second connection extends  
3 between the proxy on the intermediate node and an FTP  
4 client on the second node further comprising parsing  
5 data received from either the FTP client or the FTP  
6 server.

7 36. A method recited in claim 27, wherein the first  
8 connection extends between a file transfer protocol  
9 (FTP) client on the first node and a proxy on the  
10 intermediate node and the second connection extends  
11 between the proxy on the intermediate node and an FTP  
12 server on the second node further comprising parsing  
13 data received from either the FTP client or the FTP  
14 server.

15 37. A method recited in claim 27, wherein the first  
16 connection extends between a telnet server on the first  
17 node and a proxy on the intermediate node and the  
18 second connection extends between the proxy on the  
19 intermediate node and an telnet client on the second  
20 node further comprising parsing data received from  
21 either the telnet client or the telnet server.

22 38. A method recited in claim 27, wherein the first  
23 connection extends between a telnet client on the first  
24 node and a proxy on the intermediate node and the  
25 second connection extends between the proxy on the  
26 intermediate node and an telnet server on the second  
27 node further comprising parsing data received from  
28 either the telnet client or the telnet server.

1 39. A method recited in claim 27, wherein the first  
2 connection extends between a network news transfer  
3 protocol (NNTP) server on the first node and a proxy on  
4 the intermediate node and the second connection extends  
5 between the proxy on the intermediate node and an NNTP  
6 client on the second node further comprising parsing  
7 data received from either the NNTP client or the NNTP  
8 server.

9 40. A method recited in claim 27, wherein the first  
10 connection extends between a network news transfer  
11 protocol (NNTP) client on the first node and a proxy on  
12 the intermediate node and the second connection extends  
13 between the proxy on the intermediate node and an NNTP  
14 server on the second node further comprising parsing  
15 data received from either the NNTP client or the NNTP  
16 server.

17 41. A method recited in claim 27, wherein the first  
18 connection extends between a secure shell (SSH) server  
19 on the first node and a proxy on the intermediate node  
20 and the second connection extends between the proxy on  
21 the intermediate node and an SSH client on the second  
22 node further comprising parsing data received from  
23 either the SSH client or the SSH server.

24 42. A method recited in claim 27, wherein the first  
25 connection extends between a secure shell (SSH) client  
26 on the first node and a proxy on the intermediate node  
27 and the second connection extends between the proxy on  
28 the intermediate node and an SSH server on the second

1 node further comprising parsing data received from  
2 either the SSH client or the SSH server.

3     43. A method recited in claim 27, wherein the first  
4 connection extends between a remote shell (RSH) server  
5 on the first node and a proxy on the intermediate node  
6 and the second connection extends between the proxy on  
7 the intermediate node and an RSH client on the second  
8 node further comprising parsing data received from  
9 either the RSH client or the RSH server.

10     44. A method recited in claim 27, wherein the first  
11 connection extends between a remote shell (RSH) client  
12 on the first node and a proxy on the intermediate node  
13 and the second connection extends between the proxy on  
14 the intermediate node and an RSH server on the second  
15 node further comprising parsing data received from  
16 either the RSH client or the RSH server.

17     45. A method recited in claim 27, wherein the first  
18 connection extends between a simple mail transfer  
19 protocol (SMTP) server on the first node and a proxy on  
20 the intermediate node and the second connection extends  
21 between the proxy on the intermediate node and an SMTP  
22 client on the second node further comprising parsing  
23 data received from either the SMTP client or the SMTP  
24 server.

25     46. A method recited in claim 27, wherein the first  
26 connection extends between a simple mail transfer  
27 protocol (SMTP) client on the first node and a proxy on  
28 the intermediate node and the second connection extends

1 between the proxy on the intermediate node and an SMTP  
2 server on the second node further comprising parsing  
3 data received from either the SMTP client or the SMTP  
4 server.

5 47. A method recited in claim 27, wherein the first  
6 connection extends between a post office protocol (POP)  
7 server on the first node and a proxy on the  
8 intermediate node and the second connection extends  
9 between the proxy on the intermediate node and a POP  
10 client on the second node further comprising parsing  
11 data received from either the POP client or the POP  
12 server.

13 48. A method recited in claim 27, wherein the first  
14 connection extends between a post office protocol (POP)  
15 client on the first node and a proxy on the  
16 intermediate node and the second connection extends  
17 between the proxy on the intermediate node and a POP  
18 server on the second node further comprising parsing  
19 data received from either the POP client or the POP  
20 server.

21 49. A method recited in claim 27, wherein the first  
22 connection extends between a server sending streaming  
23 audio and or video on the first node and a proxy on the  
24 intermediate node and the second connection extends  
25 between the proxy on the intermediate node and a client  
26 receiving streaming audio or video on the second node  
27 further comprising parsing data received from either  
28 the client or the server.

1 50. A method recited in claim 37 wherein the first  
2 connection extends between a client reading streaming  
3 audio and/or video on the first node and a proxy on the  
4 intermediate node and the second connection extends  
5 between the proxy on the intermediate node and a server  
6 sending streaming audio or video on the second node  
7 further comprising parsing data received from either  
8 the client or the server.

9 51. A method recited in claim 1, further comprising:

10 forming a third connection between a third node and  
11 the intermediate node the third connection including:

12 a fifth flow originating at a fifth source flow end  
13 point on the intermediate node and terminating at a  
14 fifth destination flow end point on the third node,  
15 wherein processing at the third node associated with  
16 the fifth destination flow end point conforms to a  
17 fifth protocol, and

18 a sixth flow originating at a sixth source flow end  
19 point on the third node and terminating at a sixth  
20 destination flow end point on the intermediate node,  
21 wherein processing at the sixth node associated with  
22 the fourth source flow end point conforms to a sixth  
23 protocol,

24 splicing the fourth flow and fifth flow to form a  
25 second composite flow originating at the fourth source  
26 flow end point on the second node and terminating at  
27 the fifth destination flow end point on the third node

1 in a manner whereby the second flow and the sixth flow  
2 remain unchanged.

3 52. An article of manufacture comprising a computer  
4 usable medium having computer readable program code  
5 means embodied therein for causing the processing of  
6 network packets at an intermediate node, the computer  
7 readable program code means in said article of  
8 manufacture comprising computer readable program code  
9 means for causing a computer to effect:

10 forming a first connection between a first node and  
11 the intermediate node this first connection including:

12 a first flow originating at a first source  
13 flow end point on the first node and terminating at  
14 a first destination flow end point on the  
15 intermediate node, wherein processing at the first  
16 node associated with the first source flow end point  
17 conforms to a first protocol, and

18 a second flow originating at a second source  
19 flow end point on the intermediate node and  
20 terminating at a second destination flow end point  
21 on the first node, wherein processing at the first  
22 node associated with the second destination flow end  
23 point conforms to a second protocol;

24 forming a second connection between a second node  
25 and the intermediate node this second connection  
26 including:

1           a third flow originating at a third source  
2   flow end point on the intermediate node and  
3   terminating at a third destination flow end point on  
4   the second node, wherein processing at the second  
5   node associated with the third destination flow end  
6   point conforms to a third protocol, and

7           a fourth flow originating at a fourth source  
8   flow end point on the second node and terminating at  
9   a fourth destination flow end point on the  
10   intermediate node, wherein processing at the second  
11   node associated with the fourth source flow end  
12   point conforms to a fourth protocol,

13   such that a given flow originates at a source flow  
14   end point on a source node and terminates at a  
15   destination flow end point on a destination node and  
16   data written to the source flow end point of a given  
17   flow can subsequently be read from the destination  
18   flow end point of the given flow without traversing  
19   any intervening flow end points, and

20   splicing the first flow and third flow to form a  
21   first composite flow originating at the first source  
22   flow end point on the first node and terminating at the  
23   third destination flow end point on the second node in  
24   a manner whereby the second flow and the fourth flow  
25   remain unchanged.

26       53. An article of manufacture as recited in claim  
27   52, wherein the step of splicing the first flow and  
28   third flow to form the first composite flow allows:



1     processing at the first node associated with the  
2 first source flow end point to remain unmodified and  
3 continue to conform to the first protocol,  
4     processing at the first node associated with the  
5 second destination flow end point to remain unmodified  
6 and continue to conform to the second protocol,  
7     processing at the second node associated with the  
8 third destination flow end point to remain unmodified  
9 and continue to conform to the third protocol, and  
10    processing at the second node associated with the  
11 fourth source flow end point to remain unmodified and  
12 continue to conform to the fourth protocol.

13    54. An article of manufacture as recited in claim  
14 52, wherein the first node and the second node are the  
15 same node.

16    55, An article of manufacture as recited in claim  
17 52, wherein the first protocol and the third protocol  
18 each associate a sequence number with each byte, packet  
19 or other unit of data sent across a flow, the computer  
20 readable program code means in said article of  
21 manufacture further comprising computer readable  
22 program code means for causing a computer to effect  
23 maintaining a one to one mapping between sequence  
24 numbers associated by the first protocol with each  
25 byte, packet, or other unit of data received by the  
26 intermediate node from the first source flow end point  
27 and sequence numbers associated by the second protocol  
28 with each byte, packet, or other unit of data sent by  
29 the intermediate node to the third destination flow end  
30 point.

1     56. An article of manufacture as recited in claim  
2 52, wherein the step of splicing the first flow and  
3 third flow to form the first composite flow includes:

4     identifying a first set of packets received from the  
5 first node including all packets containing information  
6 pertaining to the first source flow end point and all  
7 packets containing information pertaining to the second  
8 destination flow end point and performing the following  
9 four steps (a), (b), (c) and (d) on each packet in  
10 this first set of packets;

11     (a) processing any information in the packet  
12     pertaining to the second flow according to the  
13     second protocol;

14     (b) replacing any information in the packet  
15     pertaining to the second flow with the corresponding  
16     information pertaining to the fourth flow;

17     (c) modifying the packet so that any information in  
18     the packet pertaining to the flow from the first  
19     source flow end point will appear to the second node  
20     as pertaining to the flow to the third destination  
21     flow end point thus establishing a correspondence  
22     between data received by the intermediate node from  
23     the first source flow end point and data sent by the  
24     intermediate node to the third destination flow end  
25     point, and

1       (d) sending each packet so processed to the second  
2       node;

3       identifying a second set of packets received from  
4       the second node including all packets containing  
5       information pertaining to the third destination flow  
6       end point and all packets containing information  
7       pertaining to the fourth source flow end point and  
8       performing the following steps (d), (e), and (f), on  
9       each packet in this second set of packets;

10      (d) identifying any information in the packet  
11      pertaining to the fourth flow and processing such  
12      information according to the fourth protocol;

13      (e) modifying any information in the packet  
14      pertaining to the flow to the third destination flow  
15      end point so the information instead pertains to the  
16      flow from the first source flow end point according to  
17      the correspondence between data received by the  
18      intermediate node from the first source flow end point  
19      and data sent by the intermediate node to the third  
20      destination flow end point established in step (b);

21      (f) sending to the first node zero or more packets  
22      containing any information resulting from step (e).

23      57. An article of manufacture as recited in claim  
24      56, wherein each of the first protocol, the second  
25      protocol, the third protocol and the fourth protocol  
26      are Transmission Control Protocol (TCP) in conjunction  
27      with either version 4 or version 6 of Internet Protocol

1 (IP) and wherein step (c) of claim 1, includes the  
2 following steps (a), (b), (c), (d), and (e):

3 (a) setting the source IP address in the IP header  
4 to the local IP address associated with the third  
5 source flow end point;

6 (b) setting the destination IP address in the IP  
7 header to the remote IP address associated with the  
8 third source flow end point;

9 (c) setting the source port number in the TCP header  
10 to the local port number associated with the third  
11 source flow end point;

12 (d) setting the destination port number in the TCP  
13 header to the remote port number associated with the  
14 third source flow end point, and

15 (e) modifying the sequence (SEQ) number in the TCP  
16 header;

17 and wherein step (b) of claim 10, includes the  
18 following steps (f), (g), (h), and (i):

19 (f) replacing the acknowledgment (ACK) number in the  
20 TCP header;

21 (g) replacing the window value in the TCP header;

22 (h) modifying or recalculating the TCP checksum in  
23 the TCP header, and

24 (i) modifying or recalculating the IP checksum in  
25 the IP header;

26 and wherein step (e) of claim 1, includes the  
27 following steps (j), (k), (l), (m), (n), (o), (p) and  
28 (q):

1 (j) setting the source IP address in the IP header  
2 to the local IP address associated with the first  
3 destination flow end point;  
4 (k) setting the destination IP address in the IP  
5 header to the remote IP address associated with the  
6 first destination flow end point;  
7 (l) setting the source port number in the TCP header  
8 to the local port number associated with the first  
9 destination flow end point;  
10 (m) setting the destination port number in the TCP  
11 header to the remote port number associated with the  
12 first destination flow end point;  
13 (n) replacing the sequence (SEQ) number in the TCP  
14 header;  
15 (o) modifying the acknowledgment (ACK) number in the  
16 TCP header;  
17 (p) modifying the TCP checksum in the TCP header,  
18 and  
19 (q) modifying the IP checksum in the IP header.

20 58. A computer program product comprising a  
21 computer usable medium having computer readable program  
22 code means embodied therein for causing the processing  
23 of network packets at an intermediate node, the  
24 computer readable program code means in said computer  
25 program product comprising computer readable program  
26 code means for causing a computer to effect:

27 forming a first connection between a first node and  
28 the intermediate node this first connection including:

1           a first flow originating at a first source  
2 flow end point on the first node and terminating at  
3 a first destination flow end point on the  
4 intermediate node, wherein processing at the first  
5 node associated with the first source flow end point  
6 conforms to a first protocol, and

7           a second flow originating at a second source  
8 flow end point on the intermediate node and  
9 terminating at a second destination flow end point  
10 on the first node, wherein processing at the first  
11 node associated with the second destination flow end  
12 point conforms to a second protocol;

13 forming a second connection between a second node  
14 and the intermediate node this second connection  
15 including:

16           a third flow originating at a third source  
17 flow end point on the intermediate node and  
18 terminating at a third destination flow end point on  
19 the second node, wherein processing at the second  
20 node associated with the third destination flow end  
21 point conforms to a third protocol, and

22           a fourth flow originating at a fourth source  
23 flow end point on the second node and terminating at  
24 a fourth destination flow end point on the  
25 intermediate node, wherein processing at the second  
26 node associated with the fourth source flow end  
27 point conforms to a fourth protocol,

1     such that a given flow originates at a source flow  
2     end point on a source node and terminates at a  
3     destination flow end point on a destination node and  
4     data written to the source flow end point of a given  
5     flow can subsequently be read from the destination  
6     flow end point of the given flow without traversing  
7     any intervening flow end points, and

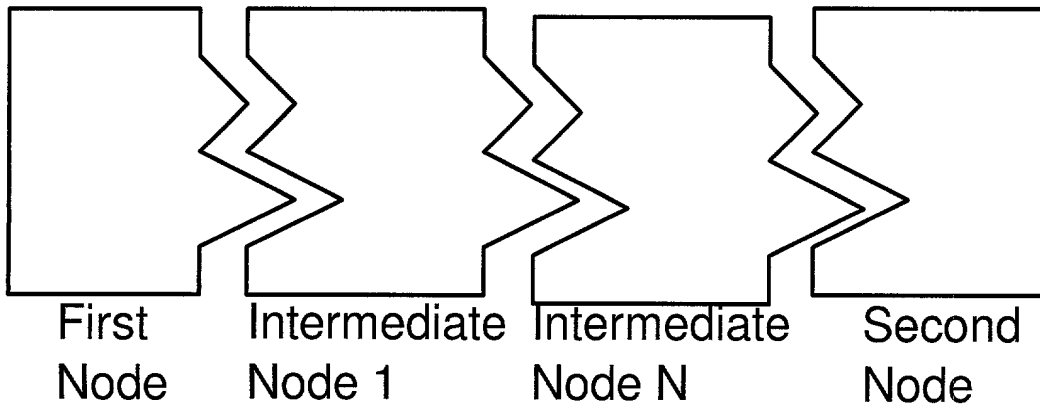
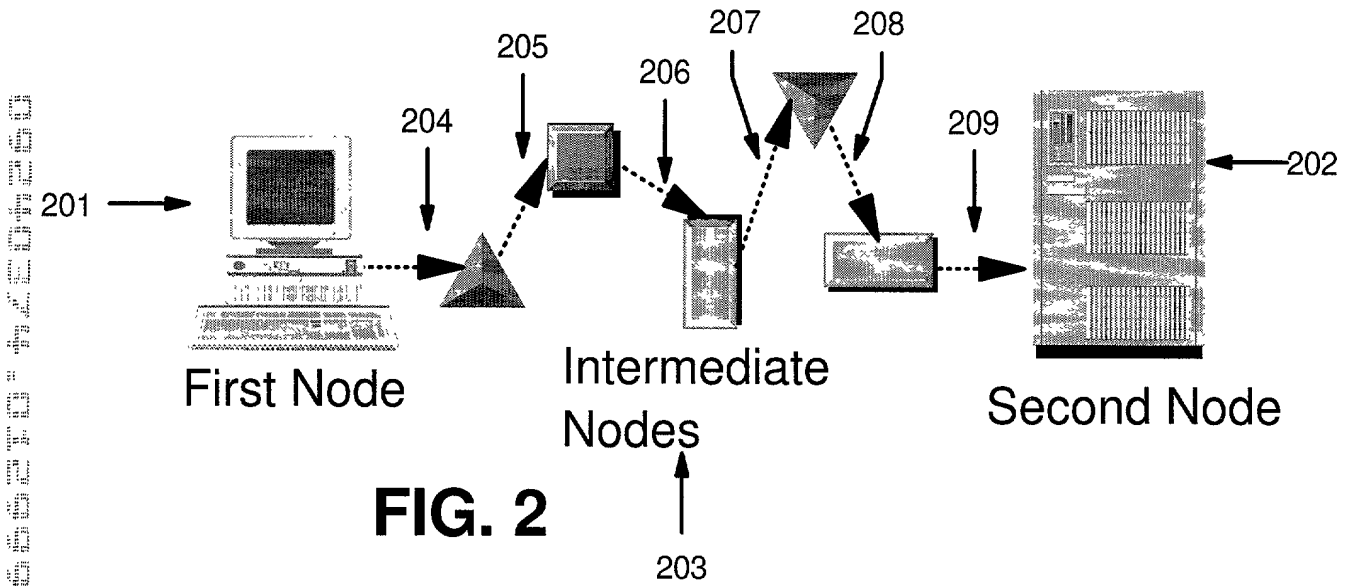
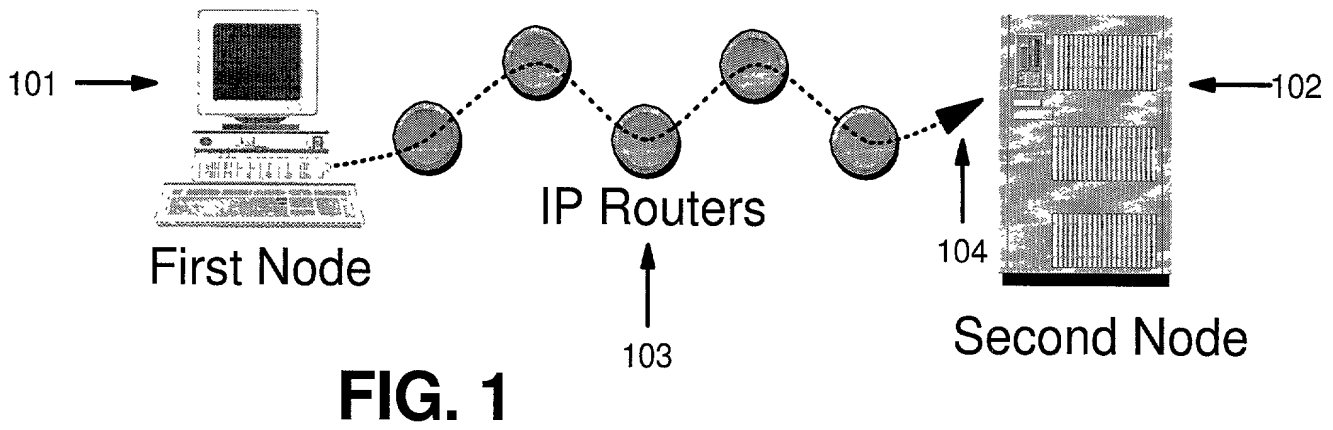
8     splicing the first flow and third flow to form a  
9     first composite flow originating at the first source  
10    flow end point on the first node and terminating at the  
11    third destination flow end point on the second node in  
12    a manner whereby the second flow and the fourth flow  
13    remain unchanged.

1     Improving Performance of Intermediate Nodes  
2                     with Flow Splicing

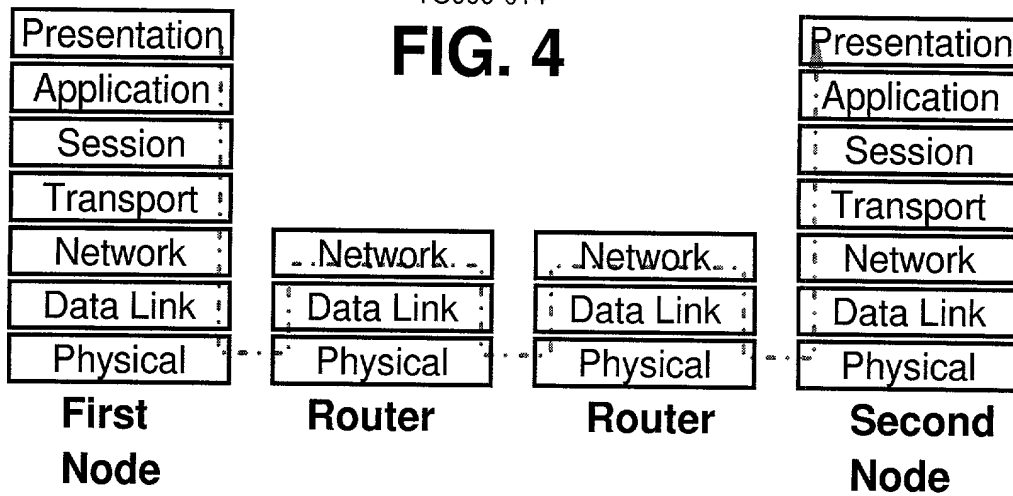
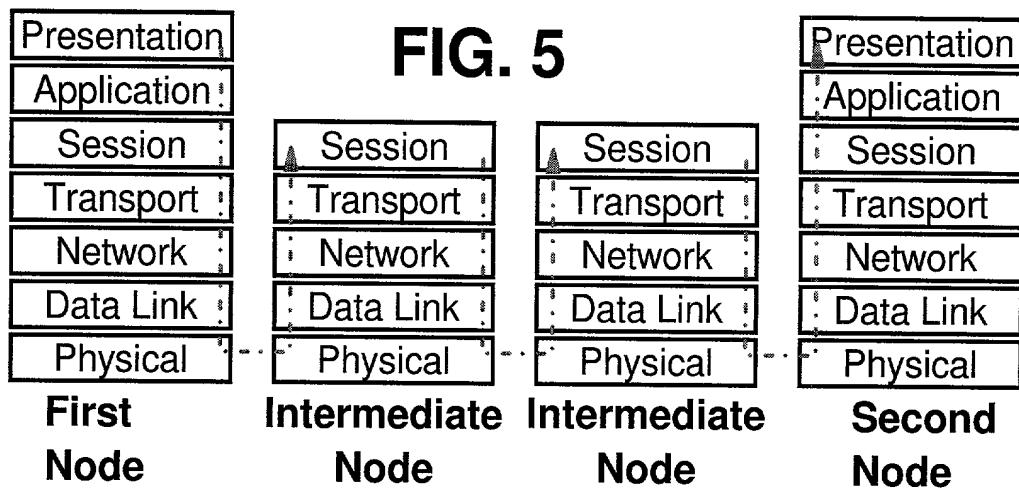
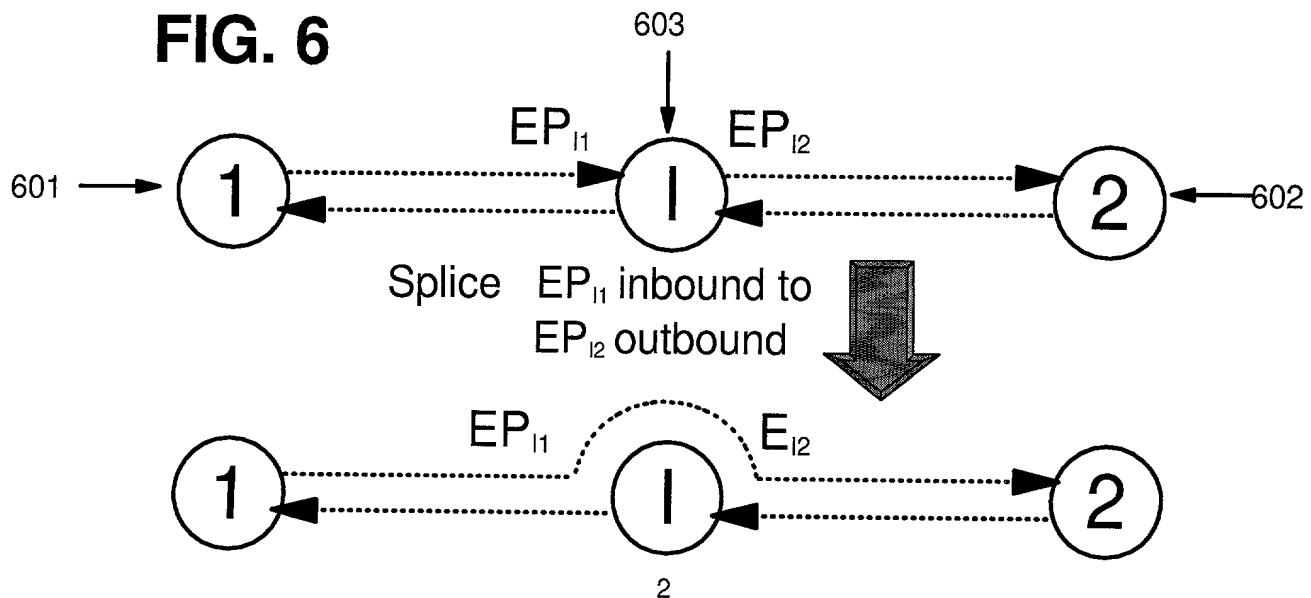
3                     ABSTRACT

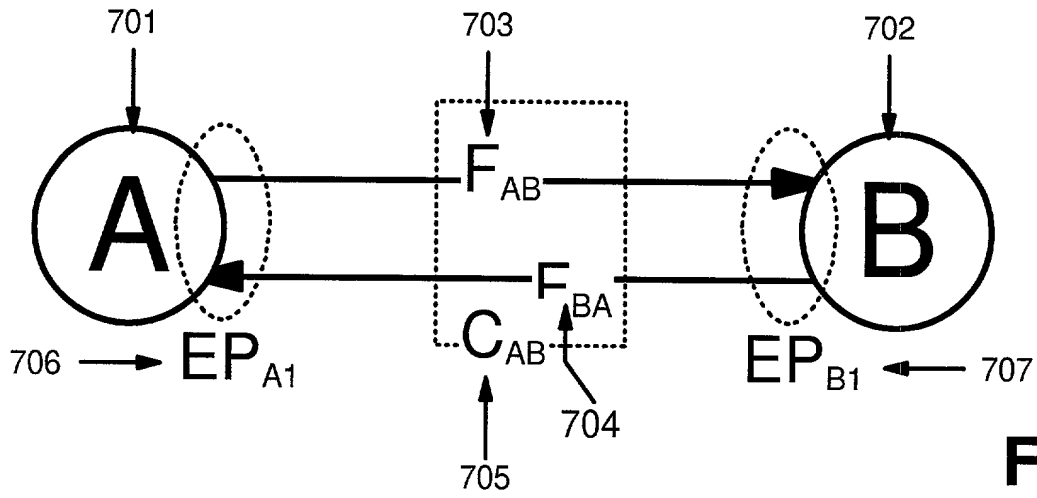
4     This invention relates to a method and apparatus for  
5     splicing a first data flow inbound to an  
6     intermediate node and second data flow outbound from  
7     the intermediate node such that the first data flow  
8     and the second data flow are transformed into a  
9     single composite data flow originating at the source  
10    of the first data flow and terminating at the  
11    destination of the second data flow. The method  
12    allows any other data flows associated with the  
13    first or second data flow, such as other data flows  
14    associated with connections that encompass either  
15    the first or second data flow, to remain unaffected  
16    by the splice. The method allows intermediate nodes  
17    in a network to influence data flow between a pair  
18    of nodes at or above the transport layer without  
19    incurring all the overhead commonly associated with  
20    transport and higher layer processing.



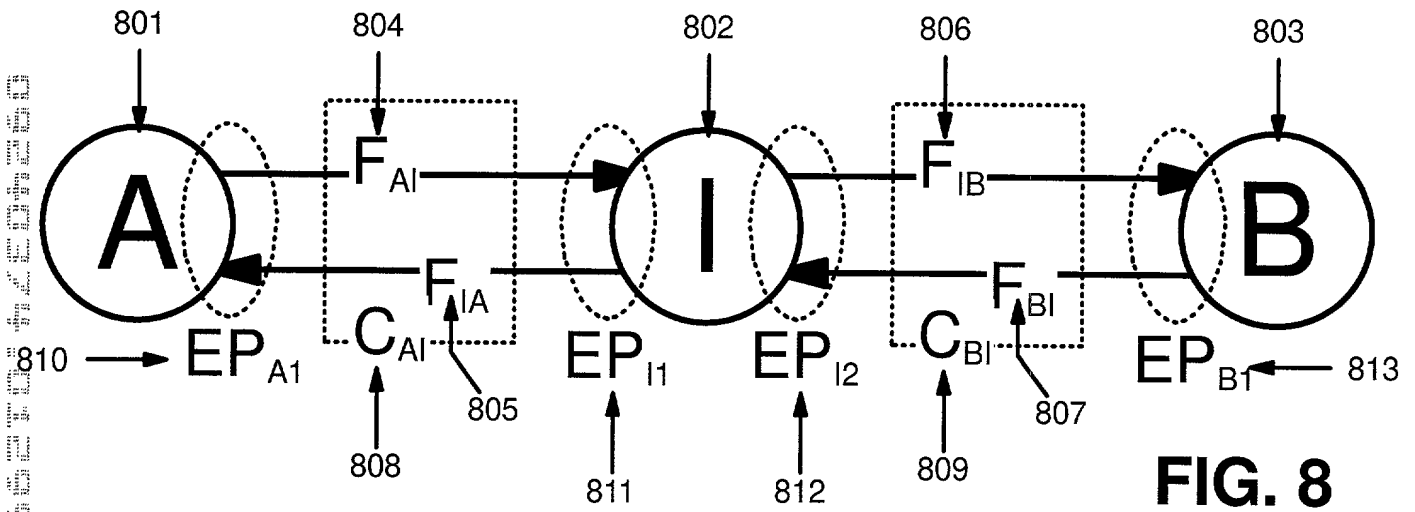


**FIG. 3**

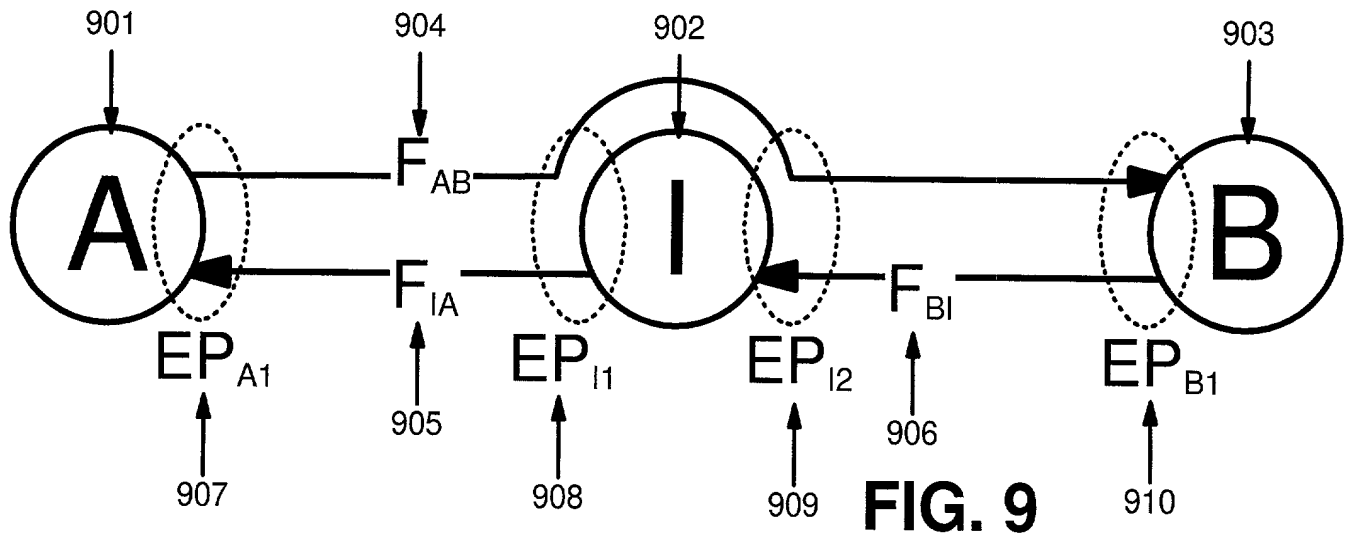
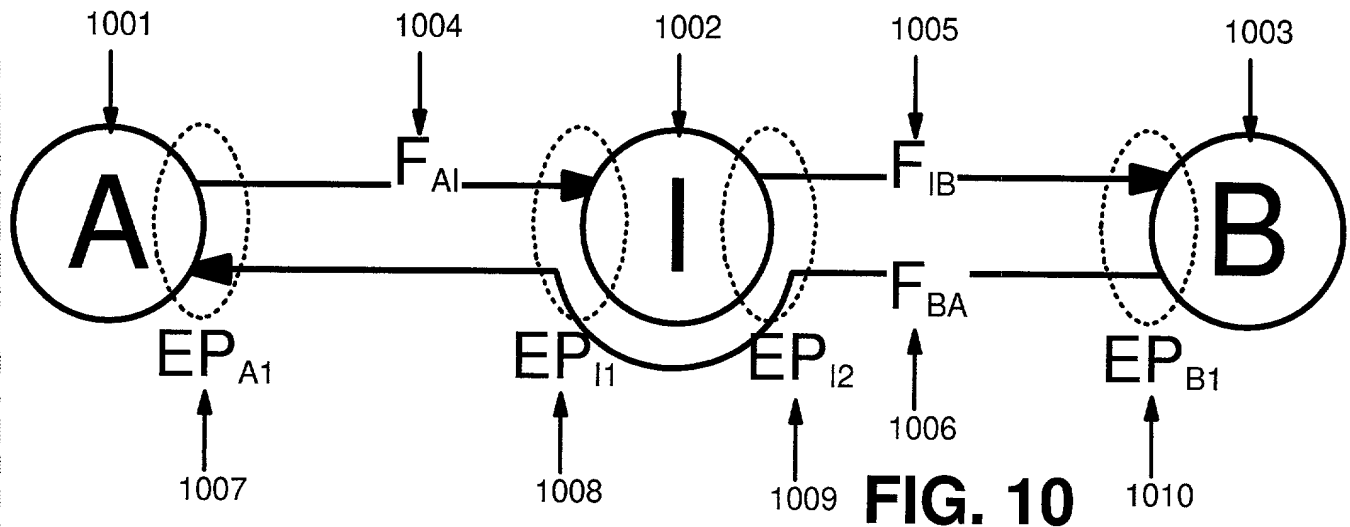
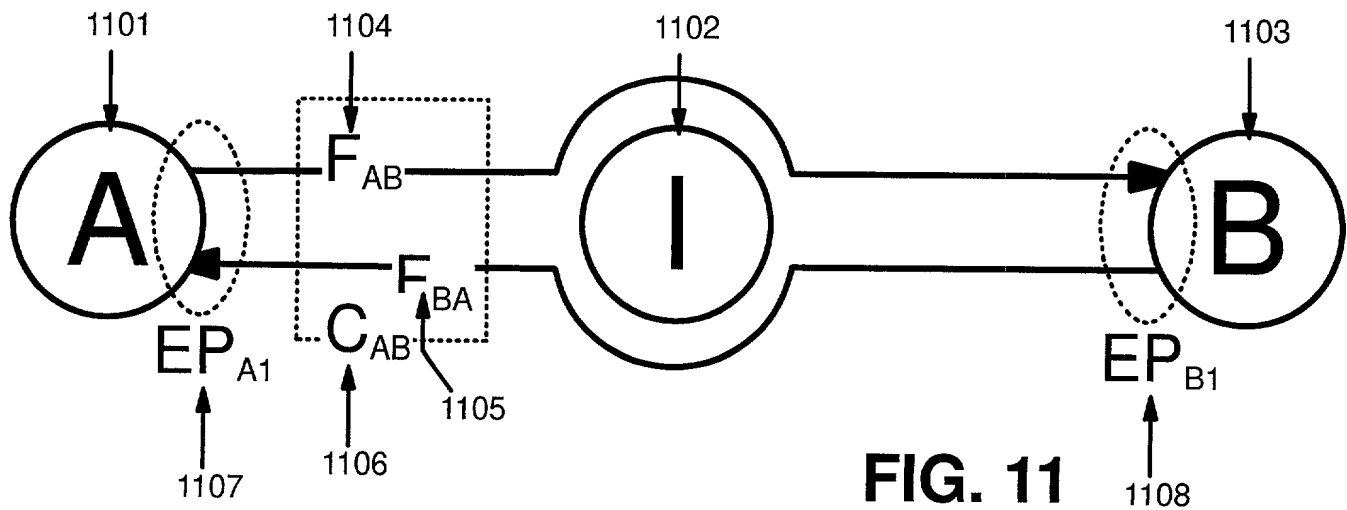
**FIG. 4****FIG. 5****FIG. 6**

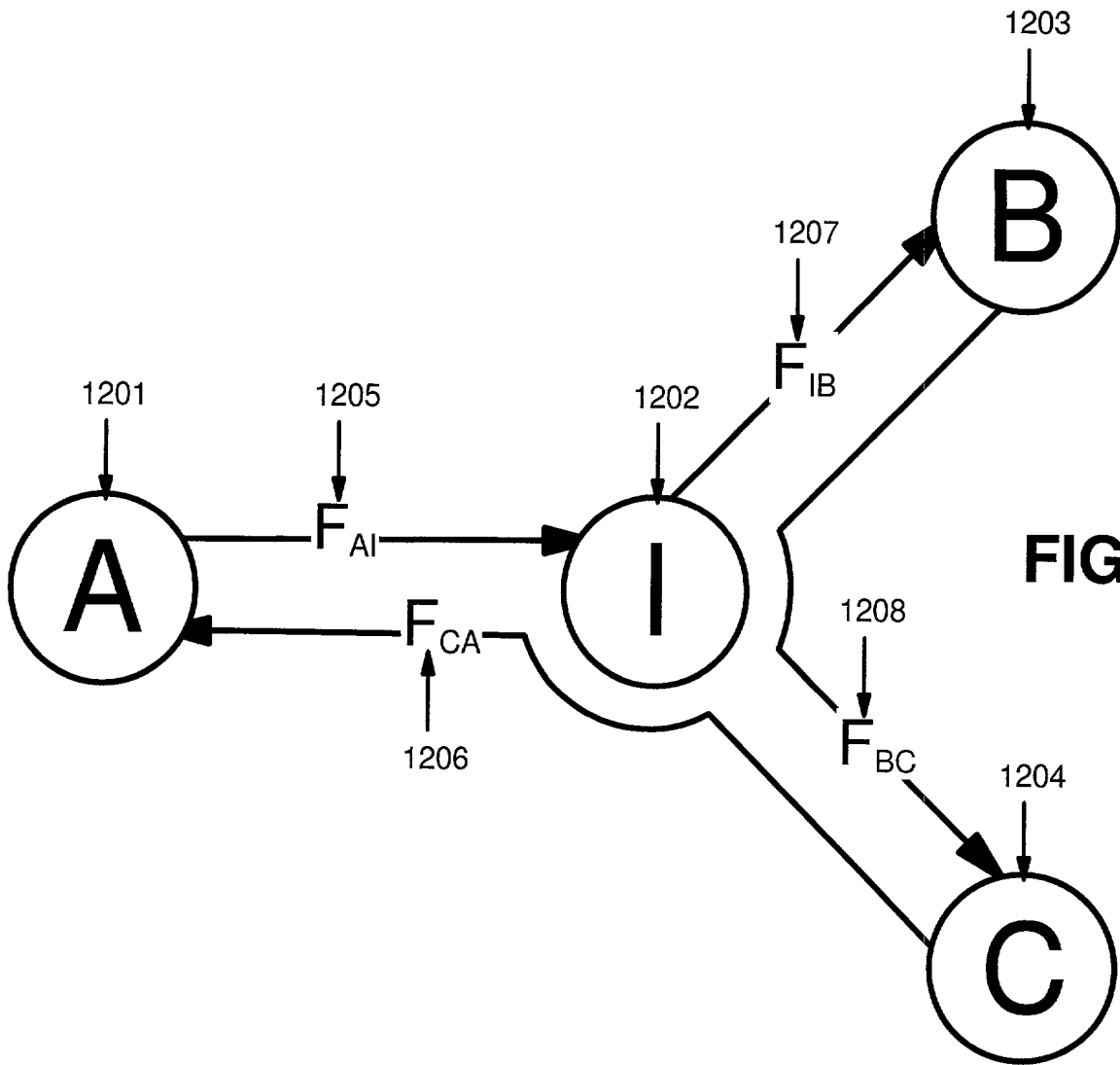


**FIG. 7**



**FIG. 8**

**FIG. 9****FIG. 10****FIG. 11**



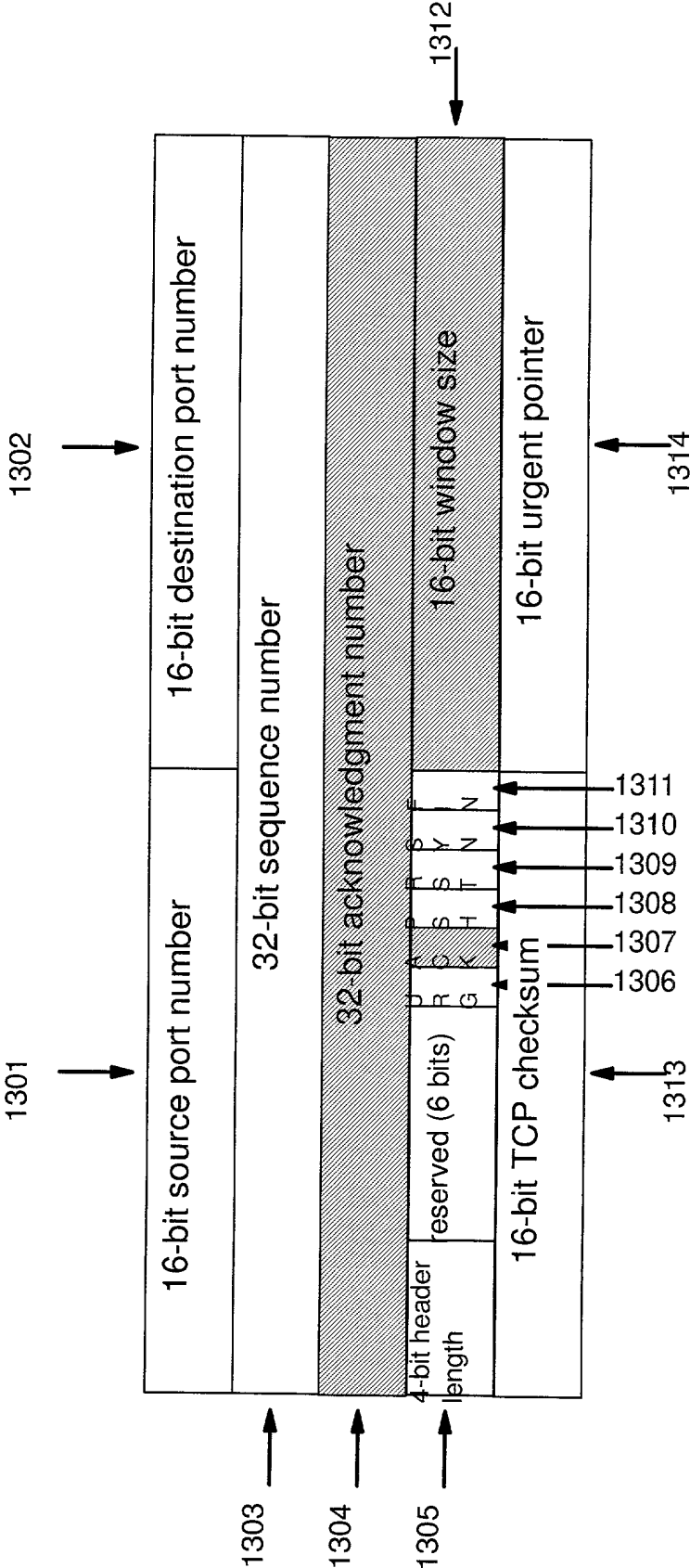
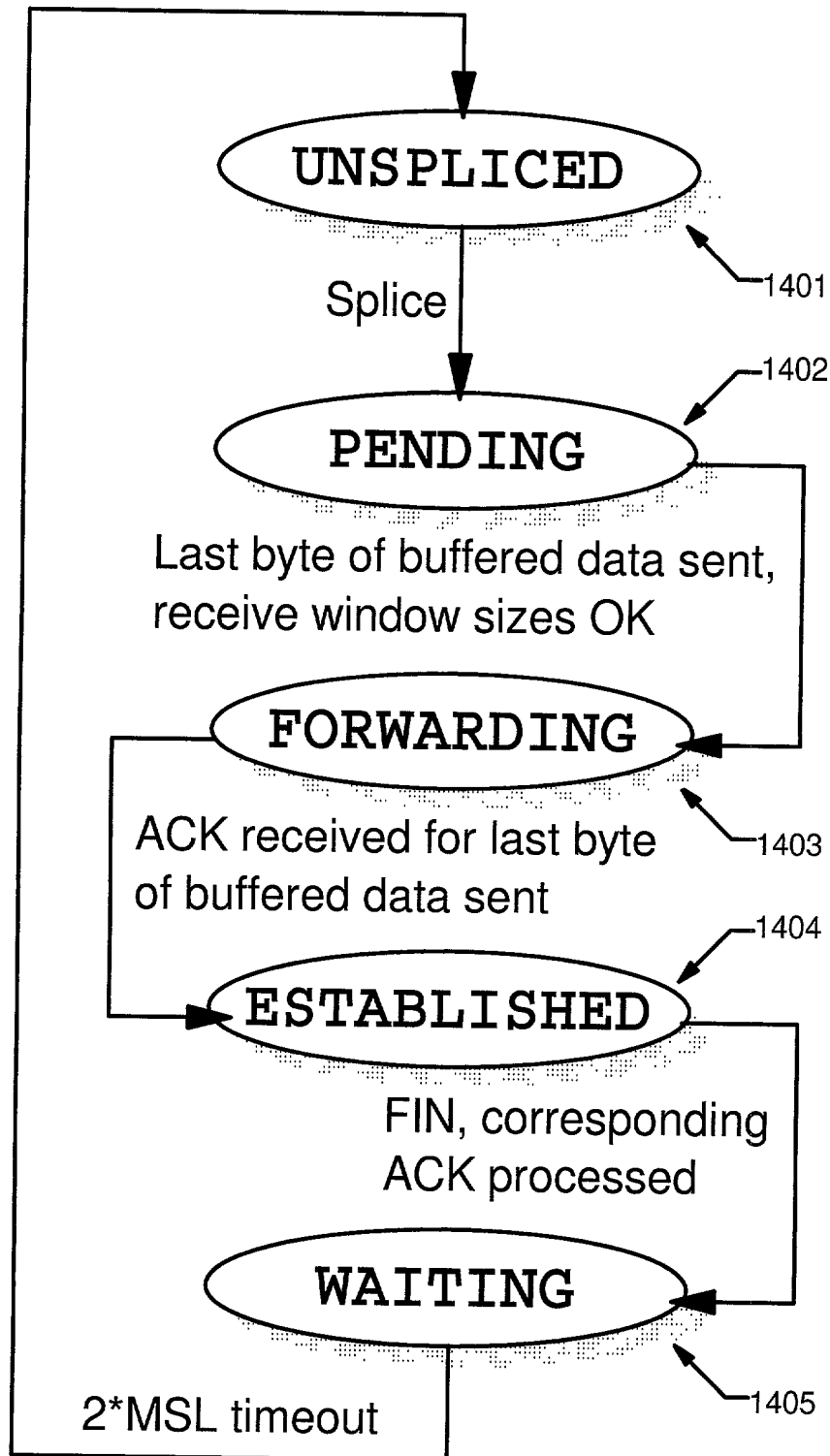


FIG. 13

**FIG. 14**

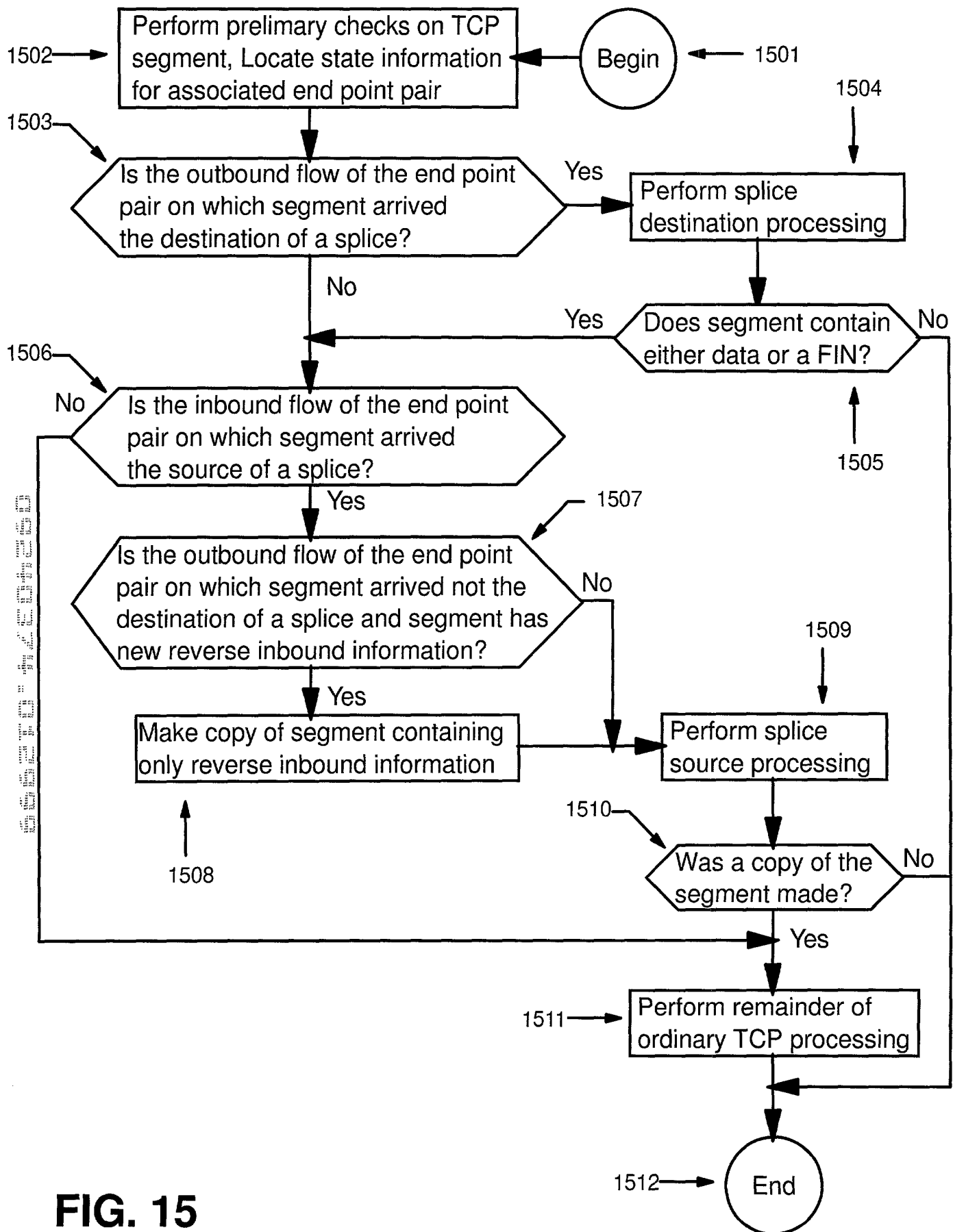


FIG. 15



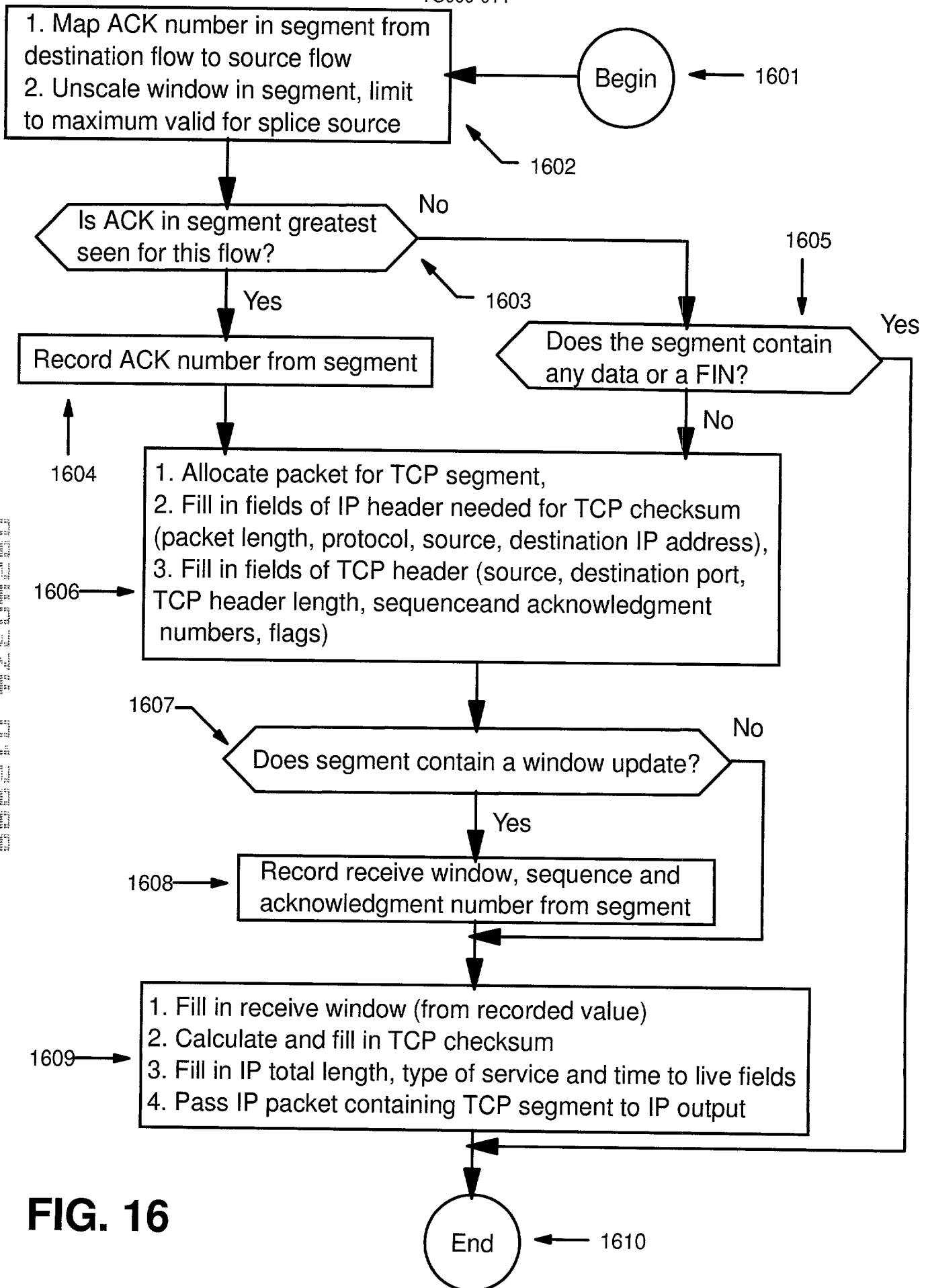
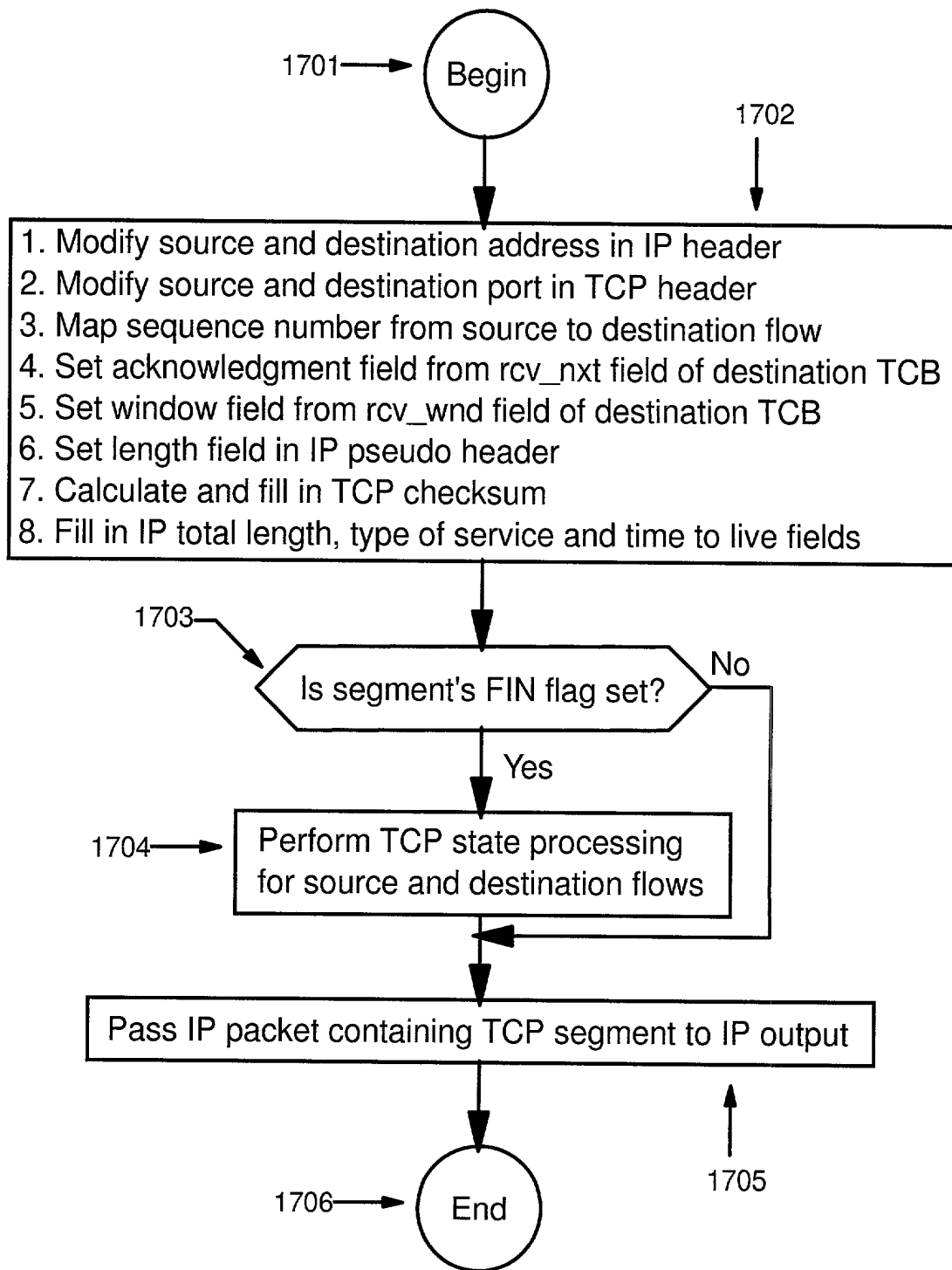


FIG. 16

**FIG. 17**

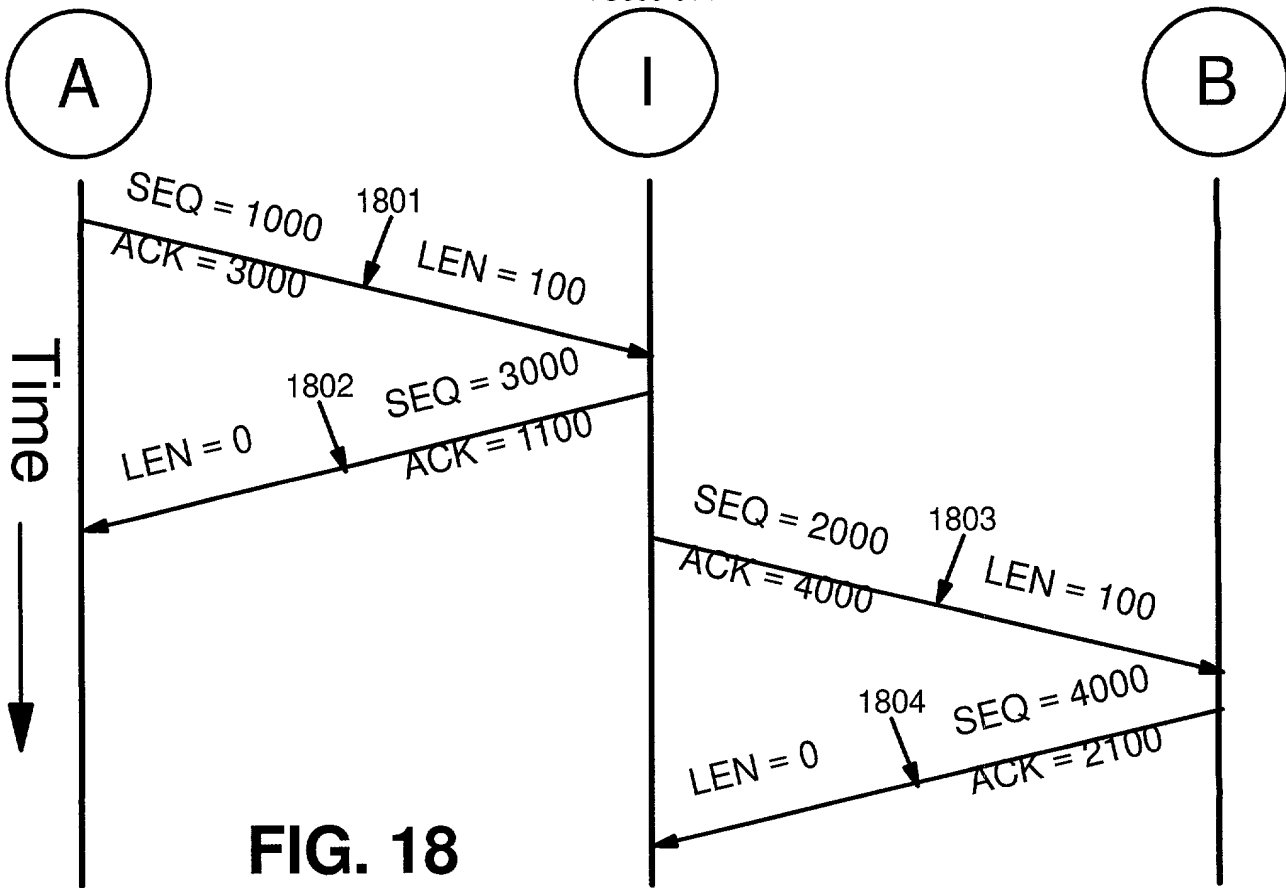


FIG. 18

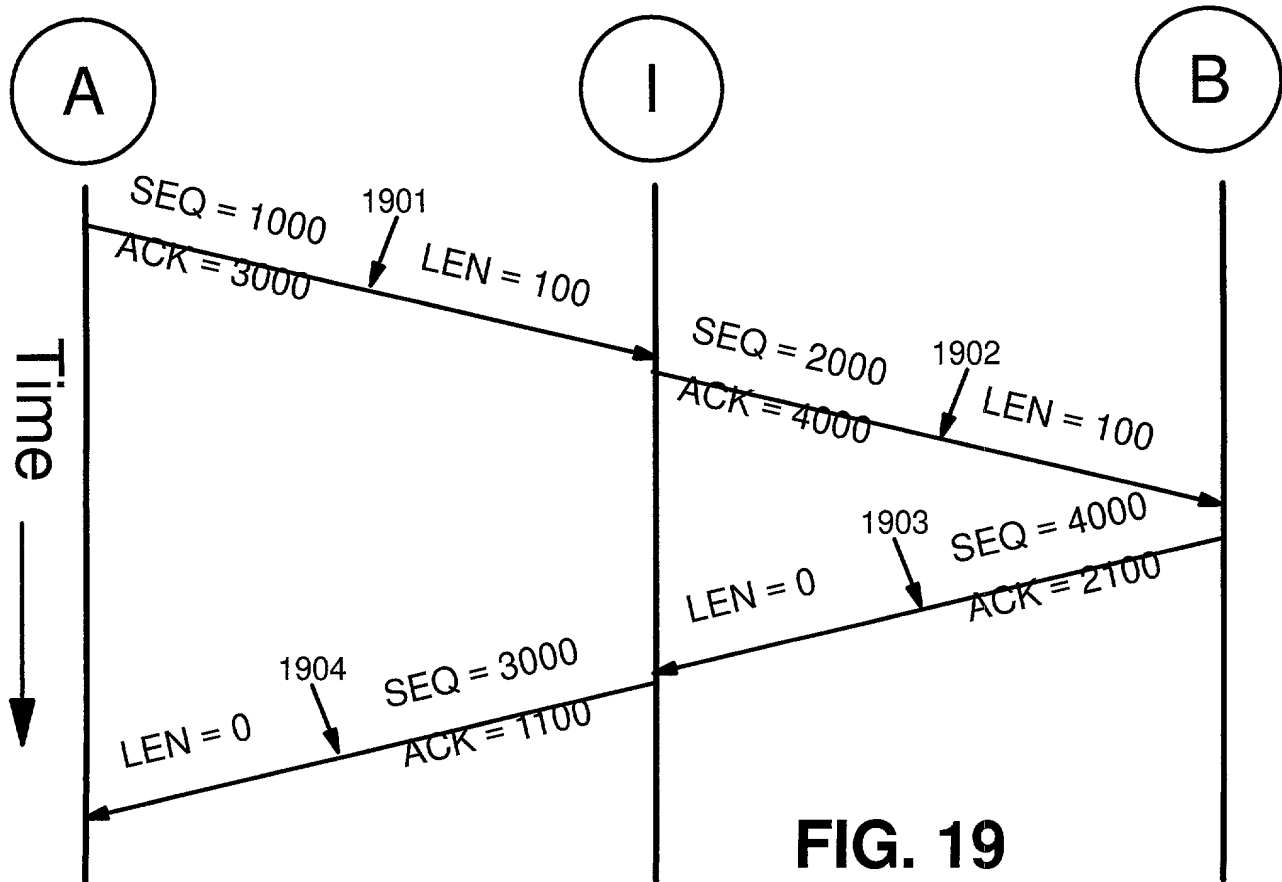


FIG. 19